Dynamické datové struktury
Binární vyhledávací strom

Prg V
4

Programování – kvinta
Verze 2.4

David Martinek, 2015–2021

Gymnázium Brno, Vídeňská, p. o.

Obsah

- ADT Binární vyhledávací strom
- BVS – operace
- Otázky a úlohy

ADT Binární vyhledávací strom

Prg V 4

- Tvoří abstraktní datový typ (ADT).
- Je to dynamická datová struktura.
  - Pro rychlé ukládání
  - a vyhledávání dat podle klíče.
- Strukturovaná data
  - Párové hodnoty: klíč + hodnota (viz algoritmy vyhledávání)
  - Např. jméno + telefonní číslo, slovo + překlad v jiném jazyce

ADT Binární vyhledávací strom

Prg V 4

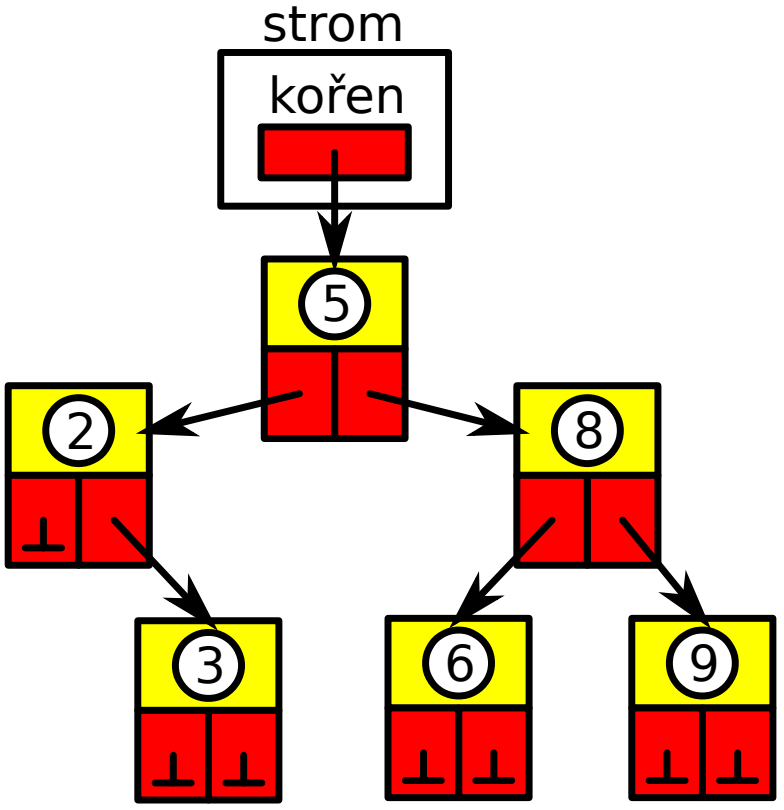
- Uzel stromu
  - Dynamická datová struktura s ukazateli na levý a pravý podstrom
  - Nese hodnotu klíče a data asociovaná s klíčem.
- Binární strom
  - Ukazatel na *kořenový uzel*
  - V naší realizaci nepřipouštíme duplicitní klíče.

ADT Binární vyhledávací strom

Prg V 4

- Rekurzivní definice
  - Strom je prázdný (kořen je NULL).
  - Strom je kořenový prvek obsahující odkazy na
    - levý podstrom (což je také strom),
    - pravý podstrom (což je také strom).
- Klíčová vlastnost
  - Klíče v levém podstromu jsou menší než klíč kořene.
  - Klíče v pravém podstromu jsou větší než klíč kořene.

ADT Binární vyhledávací strom



ADT Binární vyhledávací strom

Prg V 4

- Pojmy
  - Kořen – nejvyšší uzel
  - List – uzel s prázdnými podstromy
- Další vlastnosti
  - Výška stromu
    - Délka nejdelší cesty od kořene k listu.
  - Váha stromu
    - Počet uzlů stromu
  - Vyvážený strom
    - Pro všechny uzly platí, že váha obou podstromů se liší maximálně o jedničku.

ADT Binární vyhledávací strom

- Využití BVS
  - Pro rychlé ukládání a vyhledávání dat podle klíče.
  - Asociativní paměť
    - „Indexování“ pomocí textových řetězců nebo hodnot libovolných typů.
    - Strom se používá pro realizaci asociativního pole (v jiných jazycích).
    - Srovnej
      - pole[2] = 1.5; versus telefonniSeznam["Ondra"] = 800123777;
  - Příklady
    - Slovník, telefonní seznam, ...

Uzel stromu

Prg V 4

```
typedef struct tuzel {
  Tklic klic;   // pro zjednodušení lze Tklic ⇔ int
  Tdata data;   // pro zjednodušení výkladu vynecháno
  struct tuzel * levy; struct tuzel * pravy;
} Tuzel;


// lze i takto
typedef struct tuzel Tuzel; // předřazená specifikace
struct tuzel {
  Tklic klic; Tdata data;
  Tuzel *levy; Tuzel *pravy;
};
```

Strom – realizace

Prg V 4

```
// ADT binární vyhledávací strom
typedef struct { // obálka nad stromem
  Tuzel *koren;  // ukazatel na kořen stromu
  int vaha;      // váha stromu
  // lze přidat i další servisní informace
} Tstrom;

// Chybové kódy pro oznamování chyb
enum tchyby {EOK, EDUPKLIC, EPAMET, ENECEKANA};
```

ADT – implementační poznámka

Prg V 4

- Rozhraní vs. implementace
  - Rozhraní jsou funkce a typy určené pro uživatele ADT.
  - Implementaci tvoří funkce a typy realizující ADT „uvnitř“.
  - Koncept umožňující definovat veřejné rozhraní ADT → rozdělení podprogramů na **obalovací funkce** a **implementační funkce**

ADT – implementační poznámka

Prg V 4

- Obalovací funkce
  - Realizuje *veřejné rozhraní* pro uživatele.
  - Stará se o bezpečnost – volá implementační funkce bezpečným způsobem, vrací chybové kódy.
- Implementační funkce
  - Bývají *ukryté* před koncovým uživatelem.
  - Realizují operace ADT.
  - Výhoda – lze je měnit, aniž by to ovlivnilo programy používající ADT.

BVS – operace

- Inicializace stromu
- Průchody stromem
- Zrušení stromu
- Vložení klíče/uzlu
- Vyhledání klíče
- Odstranění klíče/uzlu
- Další průchody stromem

Inicializace stromu

Prg V 4

- Strom – obsahuje ukazatel na kořenový uzel
- Prázdný strom – obsahuje prázdný ukazatel na kořen

```
// statická inicializace
Tstrom bvsInit();

// dynamická inicializace
Tstrom* bvsInitD();
void bvsZruseni(Tstrom *s); // nezapomeň uvolnit
```

Průchody stromem

- Průchod stromem je *rekurzivní algoritmus*!
  - Jeho parametrem musí být odkaz na uzel (Tuzel*), ne na zastřešující strukturu (Tstrom*)!
    - Platí pro implementační funkci.
    - Obalovací funkce bude mít parametr typu Tstrom*.
  - Průchod měnící tvar stromu
    - Přidává nebo odebírá uzly.
    - Parametrem musí být **odkaz na uzel předávaný odkazem**!
    - Tuzel** uzel
  - Průchod neměnící tvar strom
    - Jako parametr stačí Tuzel*.

Typy průchodů stromem

- Inorder
  - inorder(levý); akceSAktuálnímUzlem(); inorder(pravý);
  - Např. výpis podle velikosti klíčů.
- Preorder
  - akceSAktuálnímUzlem(); preorder(levý); preorder(pravý);
  - Např. vyhledávání podle klíče.
- Postorder
  - postorder(levý); postorder(pravý); akceSAktuálnímUzlem();
  - Např. rušení stromu

Průchod neměnící tvar

Prg V 4

```
void _bvsInorder(Tuzel *u) {
  if (u ≠ NULL) {
    _bvsInorder(u→levy);
    printf("%d ", u→klic);
    _bvsInorder(u→pravy);
} }

void bvsInorder(Tstrom *strom) {
  _bvsInorder(strom→koren);
  free(strom);
}
```

Průchod měnící tvar

Prg V 4

```
// u je parametr typu ukazatel předávaný odkazem
void _bvsPostorder(Tuzel **u) {
  if (*u ≠ NULL) {
    _bvsPostorder(&(*u)→levy);
    _bvsPostorder(&(*u)→pravy);
    free(*u); *u = NULL; //mění uk. předaný odkazem
} }

void bvsPostorder(Tstrom *strom) {
  _bvsPostorder(&strom→koren); //mění i kořen
  free(strom);
}
```

Průchody – poznámky

Prg V 4

- Preorder a inorder mohou také měnit tvar!
  - Ale při rušení by si „podřezávaly větve“.
- Ukázka rušení s dvojitým ukazatelem demonstruje princip funkce, která mění získaný ukazatel.
  - Zrušení stromu jde realizovat i bez průběžného nastavování ukazatelů na NULL. Viz dále.

Průchod měnící tvar s fintou

Prg V 4

```
void _bvsPostorder(Tuzel **uzel) {
  Tuzel *pom = *uzel; //finta s * POZOR na konci!

  if (pom == NULL) return;
  _bvsPostorder(&pom→levy);  //zrušení levého stromu
  _bvsPostorder(&pom→pravy); //zrušení pravého stromu
  free(pom);                 // zrušení aktuálního uzlu

  *uzel = NULL;      // !!!
}
```

Poznámka k fintě při předávání ukazatelů odkazem

Prg V 4

- Měníme adresu kořenového uzlu, proto předáváme odkazem.
- Proměnná `pom` eliminuje používání dereferenčního operátoru `(*uzel)`. Ale...
  - Změna adresy v proměnné `pom` se vně funkce neprojeví, proto je nutné ji kopírovat do `*uzel`.
  - Při rekurzivním volání lze používat výraz `pom->levy` místo `(*uzel)->levy`, protože jde o složky totožné struktury.
  - Proměnné pom ani *uzel nejsou struktury, ale ukazatele na stejnou strukturu!

Poznámka k rušení stromu

- Předchozí ukázky demonstrují *princip práce s dvojitým ukazatelem*.
  - Protože to tady jde ukázat názorněji.
- Specificky rušení stromu lze dělat jednodušeji.
  - Bez rekurzivního nastavování NULL – protože se stromem už se nebude pracovat, není to potřeba.
- Při přidávání nebo odebírání jednoho uzlu už to takto zjednodušit nelze.

Zrušení stromu

- Rekurzivní průchod stromem
- Ruší se nejprve podstromy, kořen až nakonec.
  - Průchod postorder
- Protože se stromem už se nebude pracovat, je nastavování ukazatelů na NULL zbytečné.

Zrušení stromu prakticky


```
void _bvsZruseni(Tuzel *uzel) {
  if (uzel == NULL) return;
  _bvsZruseni(uzel→levy);  //zrušení levého stromu
  _bvsZruseni(uzel→pravy); //zrušení pravého stromu
  free(uzel);              // zrušení aktuálního uzlu
}

void bvsZruseni(Tstrom *strom) {
  _bvsZruseni(strom→koren); // implementace
  free(strom);         // uvolní zastřešovací strukturu
}
```

Vložení klíče

- Kombinace hledání místa pro vložení klíče (a dat) a vkládání nového uzlu.
- Nový uzel se vytváří, až když je jisté, že je to potřeba.
- V nevyvažovaném BVS se nový uzel vkládá vždy jako list.
  - Ve vyvažovaném stromu se může vkládat i jinam.

Vložení klíče

- Mohou nastat chyby
  - Operace vrací chybové kódy.
  - Uživatel této operace má povinnost tyto chybové kódy testovat.
  - Ignorování chyb vede k haváriím programu!
- Chybové stavy
  - Pokus o vložení duplicitního klíče (EDUPKLIC)
  - Není dost paměti pro alokaci nového uzlu (EPAMET)
  - Vše proběhlo v pořádku (EOK)
  - Neočekávaná chyba (ENECEKANA)

Vložení klíče

Prg V 4

```
// obalovací funkce – zajišťuje abstrakci
// pokusí se vložit do stromu nový klíč
int bvsVlozKlic(Tstrom *strom, Tklic klic) {
  int kod = _bvsVlozKlic(&strom→koren, klic);
  if (kod == EOK)
    strom→vaha++;  // váha vzrostla o 1 nový uzel

  return kod;
}
```

Vložení klíče – vytvoření uzlu

Prg V 4

```
// vrací ukazatel na nový uzel nebo NULL
Tuzel *bvsNovyUzel(Tklic klic, Tdata data) {
  Tuzel *u = malloc(sizeof(Tuzel));
  if (u =≠ NULL) { // NULL znamená, že není dost paměti
    u→klic = klic;
    u→data = data; // pro zjednodušení jinde chybí ...
    u→levy = u→pravy = NULL; // nezapomenout!
  }
  return u;
}
```

Vložení klíče – rekurzivní implementace

Prg V 4

```
// koren je parametr typu ukazatel předávaný odkazem
int _bvsVlozKlic(Tuzel **koren, Tklic klic)
{
  Tuzel *pom = *koren;//pomůcka pro eliminaci *, POZOR!!!

  if (pom == NULL) { // vkládáme do prázdného stromu
    if ((pom = bvsNovyUzel(klic)) == NULL) return EPAMET;
    *koren = pom; // NUTNÉ!!! pom je lokální
    return EOK;
  }
  // pokračování dále …
```

Vložení klíče – rekurzivní implementace

Prg V 4

```
  // … pokračování
  // nevkládáme náhodou duplicitní klíč?
  if (isEqual(klic, pom→klic)) return EDUPKLIC;

  // teď je jasné, že klíč patří vlevo nebo vpravo
  if (isLess(klic, pom→klic))
    return _bvsVlozKlic(&pom→levy, klic); // odkazem!
  else
    return _bvsVlozKlic(&pom→pravy, klic); // odkazem!
}
```

Vyhledání klíče

- Parametry algoritmu
  - Ukazatel na kořen stromu či podstromu
  - Vyhledávaný klíč
- Výsledek algoritmu
  - Ukazatel na uzel s nalezeným klíčem
  - NULL, když klíč nebyl nalezen
  - Při implementaci s páry klíč-data lze vracet rovnou data asociovaná s klíčem – při reálné implementaci vhodnější.

Vyhledání klíče

Prg V 4

- Rekurzivní algoritmus (preorder)
  - Je ukazatel na kořen roven NULL?
    - Klíč nenalezen, vrať NULL.
  - Je klíč shodný s klíčem kořene?
    - Klíč nalezen, vrať ukazatel na kořen.
  - Je klíč menší než klíč kořene?
    - Vrať výsledek hledání v levém podstromu.
  - Je klíč větší než klíč kořene?
    - Vrať výsledek hledání v pravém podstromu.

Vyhledání klíče

Prg V 4

- Operace nemění žádné uzly ⇒ ukazatel na uzel není potřeba předávat odkazem.

```
// obalovací funkce – hledá klíč ve stromu
Tuzel *bvsNajdiKlic(Tstrom *strom, Tklic klic) {
  return _bvsNajdiKlic(strom→koren, klic);
}

// implementační funkce, rekurzivně hledá klíč
Tuzel *_bvsNajdiKlic(Tuzel *koren, Tklic klic);
```

Odstranění klíče/uzlu

- Kombinace vyhledání klíče a odstranění uzlu.
- I po odstranění uzlu musí strom zachovávat vlastnosti binárního vyhledávacího stromu.
- Operace modifikuje strom, proto se ukazatel na kořen předává odkazem.

Algoritmus odstranění klíče

- Vstupem je ukazatel na kořenový uzel a klíč.

1. Rekurzivně *najdi uzel* s odpovídajícím klíčem.
2. Pokud nebyl nalezen, operace nic neruší, končí.
3. *Nalezený uzel zruš* algoritmem pro odstranění uzlu.
   - Ukazatel na tento uzel se předává do funkce odkazem, protože tento podstrom se bude měnit.

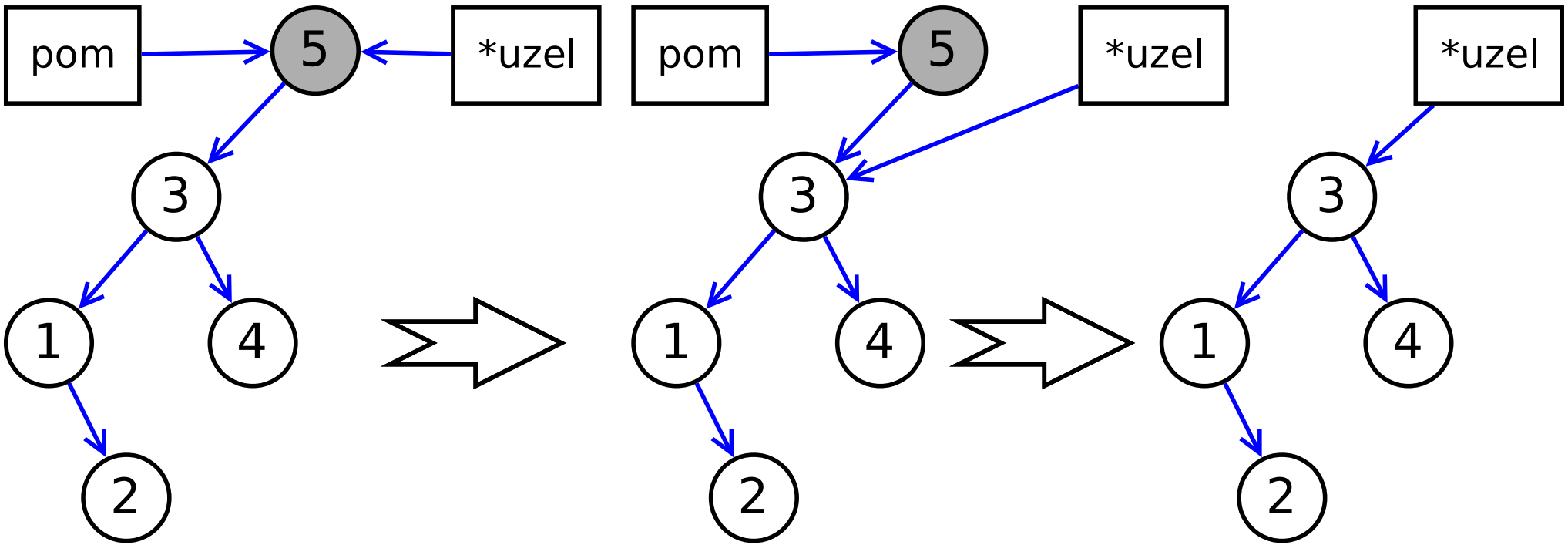
Algoritmus odstranění uzlu

Prg V 4

- Vstupem je ukazatel na uzel určený ke zrušení.

1. Je ukazatel na rušený uzel prázdný?
   - Pak není co rušit, skonči.
2. Je levý podstrom rušeného uzlu prázdný?
   - Ukazatel na aktuální uzel nastav na ukazatel na pravý podstrom.
   - Zruš nyní již přemostěný uzel.
3. Je pravý podstrom rušeného uzlu prázdný?
   - Postupuj analogicky jako při předchozím kroku.

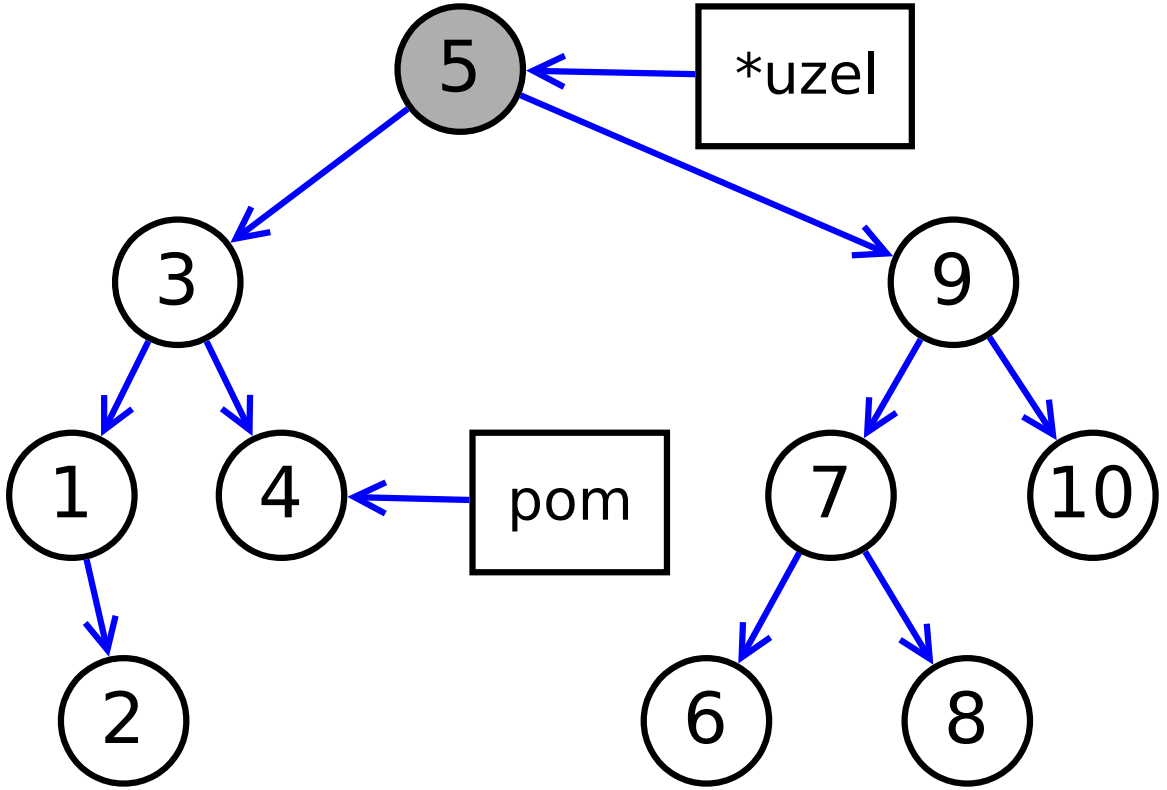
Algoritmus odstranění uzlu



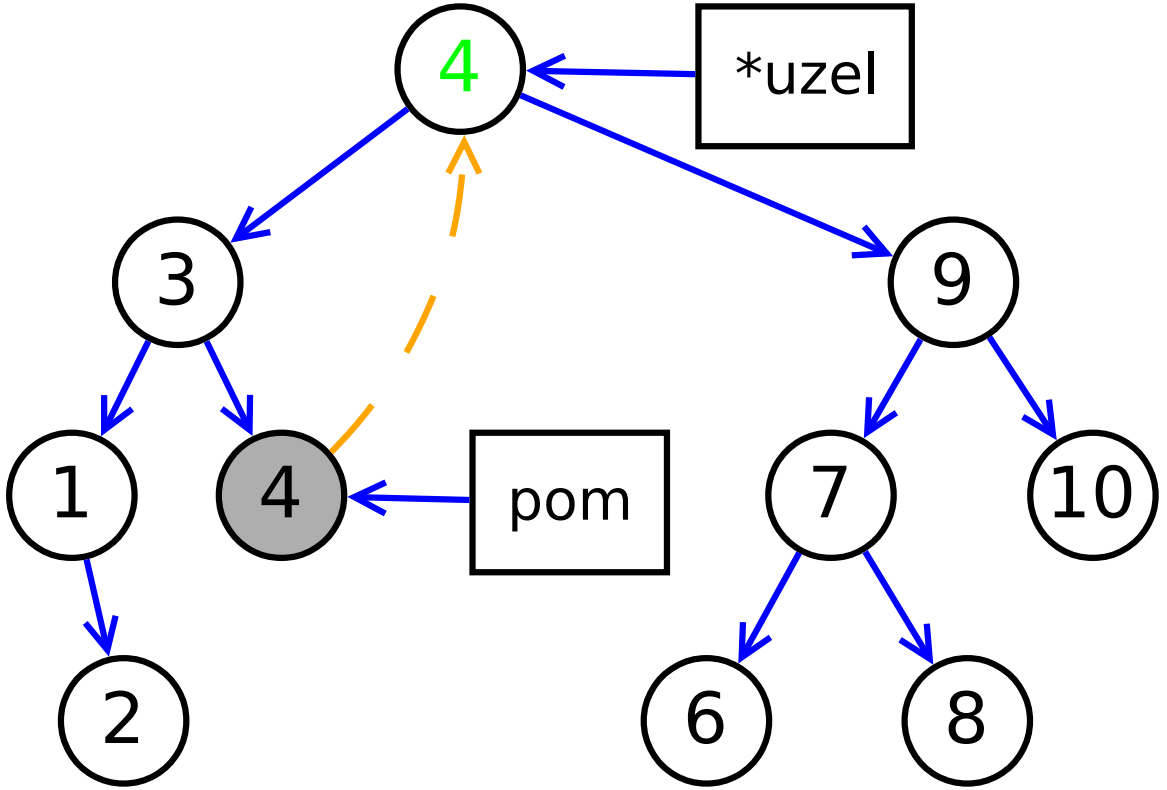
Algoritmus odstranění uzlu

- Idea před pokračováním algoritmu: uzel má oba podstromy, proto
  - Najdeme největší klíč v levém podstromu.
  - Nachází se v nejpravějším uzlu levého podstromu.
  - Tento klíč (+ data) zkopírujeme do uzlu, který jsme chtěli původně zrušit.
  - Místo uzlu původně určeného k rušení nyní zrušíme tento nejpravější uzel v levém podstromu.
  - Tento postup zajistí zachování vlastností binárního vyhledávacího stromu.

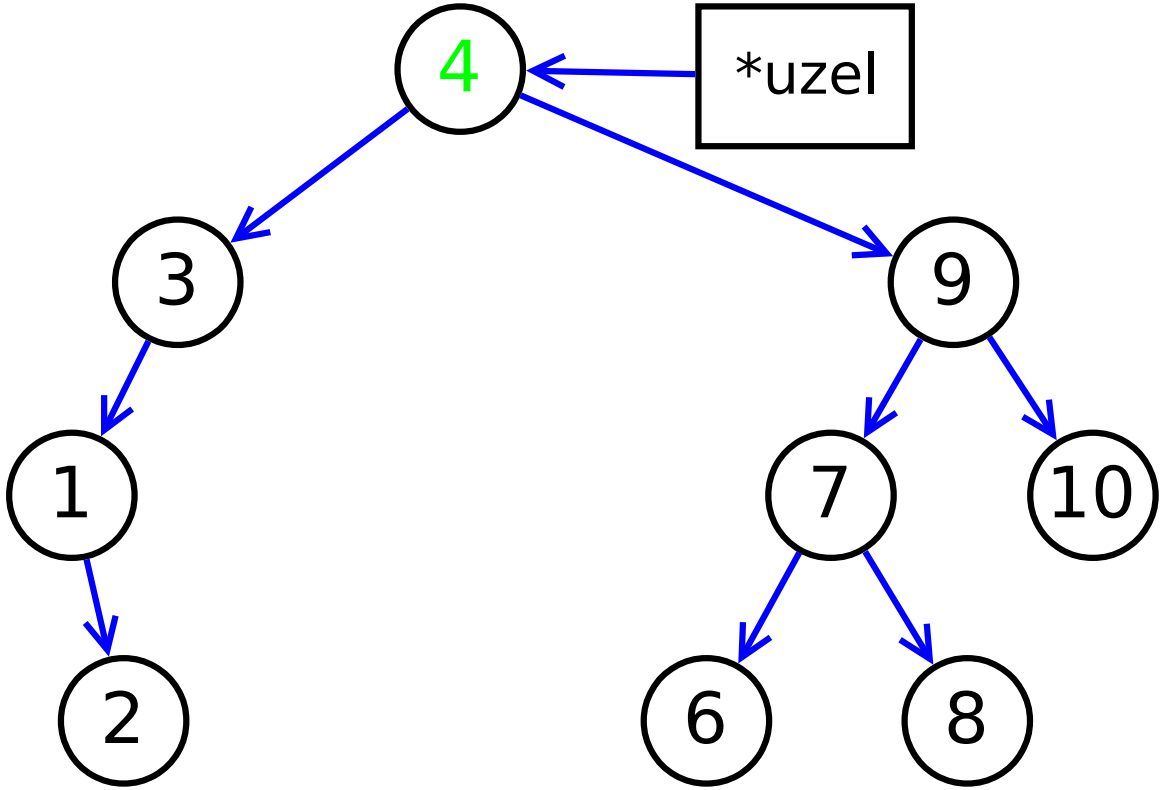
Algoritmus odstranění uzlu



Algoritmus odstranění uzlu



Algoritmus odstranění uzlu



Algoritmus odstranění uzlu

Prg V 4

4. Je pravý podstrom levého podstromu rušeného uzlu prázdný?
   - Kopíruj klíč (a data) z levého uzlu do aktuálního uzlu.
   - Rekurzivně zruš levý uzel a skonči.
5. Jinak
   - Nastav pomocný ukazatel pom na levý podstrom aktuálního uzlu.
   - Posouvej pomocný ukazatel pom stále doprava, dokud nebude ukazovat na nejpravější uzel.
   - Kopíruj klíč (a data) z uzlu na něhož ukazuje pom do původního uzlu určeného ke zrušení.
   - Rekurzivně zruš uzel odkazovaný pom a skonči.

Odstranění klíče

Prg V 4

```
// obalovací funkce - hledá klíč ve stromu
int bvsZrusKlic(Tstrom *strom, Tklic klic) {
  return _bvsZrusKlic(&strom→koren, klic);
}
// Rekurzivně hledá uzel s klíčem, který pak ruší.
// Volá funkci _bvsZrusUzel.
int _bvsZrusKlic(Tuzel **koren, Tklic klic);

// Zruší zadaný uzel
int _bvsZrusUzel(Tuzel **uzel);
```

Odstranění klíče

Prg V 4

```
// Zruší zadaný uzel, vrací chybový kód
int _bvsZrusUzel(Tuzel **uzel) {
  Tuzel *pom = *uzel;
  if (pom == NULL) return ENECEKANA;

  if (pom->levy == NULL) {
    *uzel = pom->pravy; free(pom);  // !!!
    return EOK;
  }
  if (pom->pravy == NULL) // analogicky
```

Odstranění klíče

Prg V 4

```
  if (pom→levy→pravy == NULL) {
    pom→klic = pom→levy→klic;   //+ totéž pro data
    return _bvsZrusUzel(&pom→levy);
  }
  pom = pom→levy;
  while (pom→pravy→pravy ≠ NULL) pom=pom→pravy;

  (*uzel)→klic = pom→pravy→klic;//+ totéž pro data
  return _bvsZrusUzel(&pom→pravy);
}
```

Další průchody stromem

Prg V 4

- Průchody do hloubky
  - Průchod všemi uzly stromu
  - První cesta vede od kořene až k listu
  - Preorder, Inorder, Postorder
  - Inorder ve vyhledávacím stromu – zpracovává uzly podle pořadí daného klíči
- Průchody do šířky
  - Průchod uzly do zvolené hloubky
  - Průchod po uzlech na zvolené úrovni

Průchod s obecnou akcí

- Průchody jsou častá operace.
- Často je potřeba nad uzly provádět různé aktivity.
- Předchozí implementace průchodů je málo obecná.
- Řešení: průchod s obecnou akcí nad uzlem
  - realizace pomocí typu ukazatel na funkci

Průchod Preorder s obecnou akcí

Prg V 4

```
typedef void (*TFakce)(Tuzel *);
void _bvsPreorder(Tuzel *u, TFakce akce) {
  if (u ≠ NULL) {
    akce(u);
    _bvsInorder(u→levy);
    _bvsInorder(u→pravy);
} }

void bvsPreorder(Tstrom *strom, TFakce akce) {
  _bvsPreorder(strom→koren, akce);
}
```

Průchod Preorder s obecnou akcí

Prg V 4

```
void tiskniKlic(Tuzel *u) {
  printf("%d ", u→klic); // pokud Tklic je int
}
void tiskniData(Tuzel *u) {
  printf("%d ", u→data); // pokud Tdata je int
}
int main(void) {
  bvsPreorder(strom→koren, tiskniKlic);
  bvsPreorder(strom→koren, tiskniData);
}
```

Průchod do zvolené hloubky

- Parametrem rekurzivní funkce musí být počítadlo hloubky.
- Při každém rekurzivním zanoření se musí měnit o jedničku.
- Při vynoření z rekurze na zásobníku zůstává původní hodnota počítadla.

Průchod Postorder do zvolené hloubky

Prg V 4

```
void _bvsHInorder(Tuzel *u, unsigned int hloubka) {
  if (u == NULL) return;
  if (hloubka > 0) {
    _bvsInorder(u→levy, hloubka - 1);
    _bvsInorder(u→pravy, hloubka -1);
  }
  printf("%d ", u→klic);
}
void bvsHInorder(Tstrom*strom, unsigned int hloubka) {
  _bvsInorder(strom→koren, hloubka);
}
```

Průchod po uzlech na zvolené úrovni

{Prg V 4}

- Rekurzivní průchod (preorder)
- Akce nad uzlem se vyvolá jen u uzlů zadané úrovně (hloubky).
- Pokud je vyžadován často, lze zefektivnit modifikací datové struktury → provázání uzlů na stejných úrovních pomocí ukazatelů → průchod lineárním seznamem.

Průchod po uzlech na zvolené úrovni

Prg V 4

```
void _bvsUroven(Tuzel *u, unsigned int hloubka,
                TFakce akce) {
  if (u ≠ NULL)
    if (hloubka == 0)
      akce(u);
    else {
      _bvsUroven(u→levy, hloubka-1, akce);
      _bvsUroven(u→pravy, hloubka-1, akce);
}   }
```

Otázky a úlohy

Prg V 4

- Popiš rekurzivní definici BVS.
- Jaká je klíčová vlastnost BVS?
- K čemu se BVS používá?
- Popiš a vysvětli průchod inorder/preorder/postorder. K čemu je to dobré? Uveď příklad použití.
- Jak se liší implementace průchodů, které ne/mění tvar stromu?
- Proč průchod měnící tvar stromu potřebuje jako parametr dvojitý ukazatel?

Otázky a úlohy

Prg V 4

- Deklaruj datový typ pro realizaci binárního vyhledávacího stromu, v němž budou klíč i data textové řetězce.
- Slovně popiš a na papír načrtni algoritmus odstranění zadaného klíče ze stromu.
- V jazyce C napiš funkci, která vytiskne všechny klíče binárního vyhledávacího stromu v pořadí
  - od nejmenšího po největšího,
  - od největšího po nejmenšího.

Otázky a úlohy

- Popiš algoritmus vložení klíče do BVS.
- Popiš algoritmus zrušení stromu.
- Popiš algoritmus vyhledání klíče v BVS.

Otázky a úlohy

- Pomocí BVS realizuj jednoduchý překladový slovník z angličtiny do češtiny.
  - Anglické slovo je klíč. Klíče porovnávej pomocí strcmp().
  - České slovo tvoří data asociovaná s klíčem.
  - Naplň slovník dvojicemi zapsanými v souboru.
  - Přelož zadanou anglickou větu.