B0B99PRPA – Procedurální programování

Stromy.

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Stromy

  Stromy

- Část 2 – Preprocesor

  Preprocesor

Část I

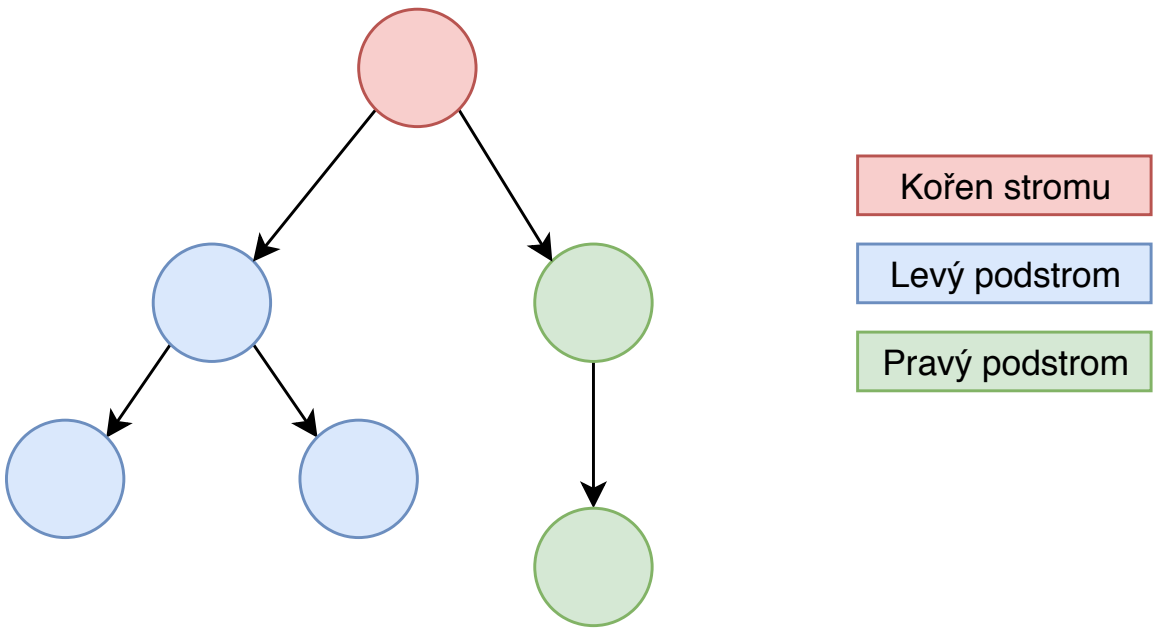
Stromy

I. Stromy

Stromy

Lineární a nelineární spojové struktury

- Spojové seznamy predstavují lineární spojovou strukturu
  - Každý prvek má nejvýše jednoho následníka
- Nelineární spojové struktury (např. stromy)
  - Každý prvek může mít více následníků
- **Binární strom** – každý prvek (uzel) má nejvýše dva následníky



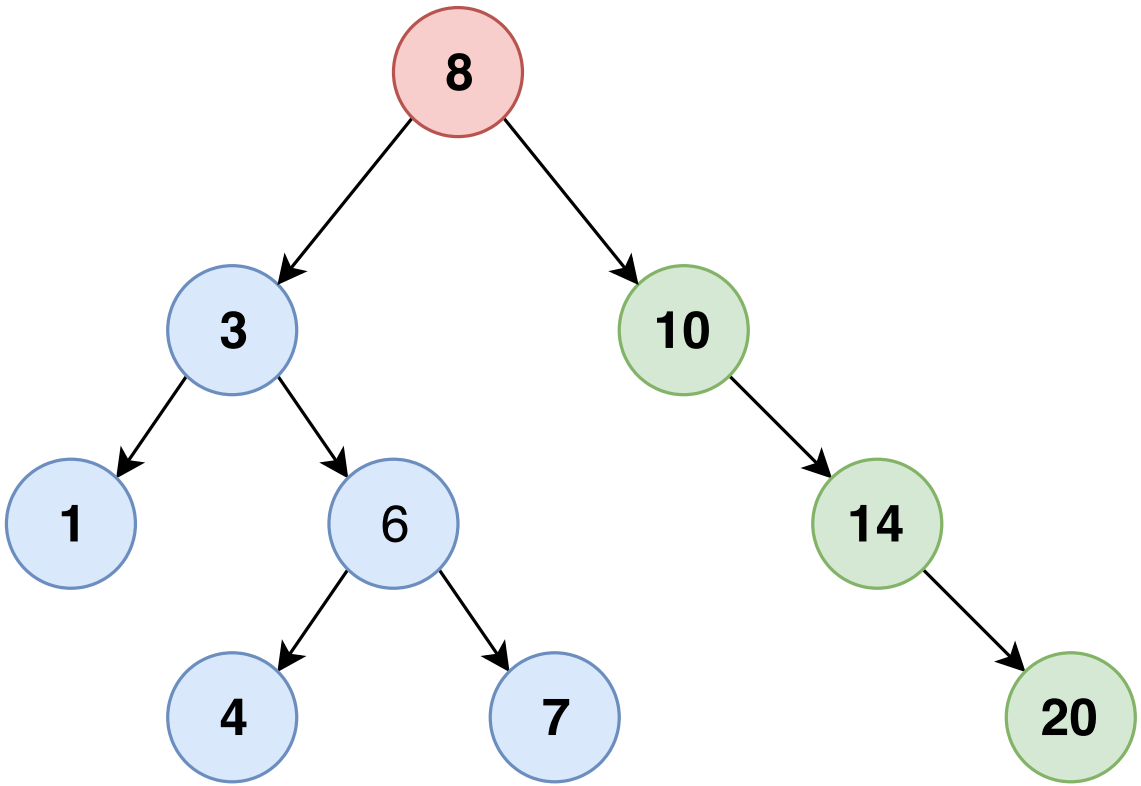
Binární strom

- uvažujme datové položky uzlu stromu jako hodnoty typu int
- uzel stromu reprezentujeme strukturou node_t
- strom je pak reprezentován kořenem stromu, ze kterého máme přístup k jednotlivým uzlům
  - potomci left a right a jejich potomci

```
typedef struct node {
  int value;
  struct node *left;
  struct node *right;
} node_t;
node_t *tree;
```

Příklad – Binární vyhledávací strom

- Binární vyhledávací strom – Binary Search Tree (BST)
- Pro každý prvek (uzel) platí
  - hodnota (value) potomka vlevo je menší (nebo NULL)
  - hodnota potomka vpravo je větší (nebo NULL)



Binární vyhledávací strom – vložení prvku 1/2

- pro vložení prvku nejprve dynamicky alokujeme uzel

```
static node_t* newNode(int value)
{
  node_t *node= (node_t*)malloc(sizeof(node_t));
  node->value = value;
  node->left = node->right = NULL;
  return node;
}
```

Binární vyhledávací strom – vložení prvku 2/2

- vložení alokovaného prvku – využijeme rekurze a vkládáme na první volné vhodné místo, splňující podmínku BST

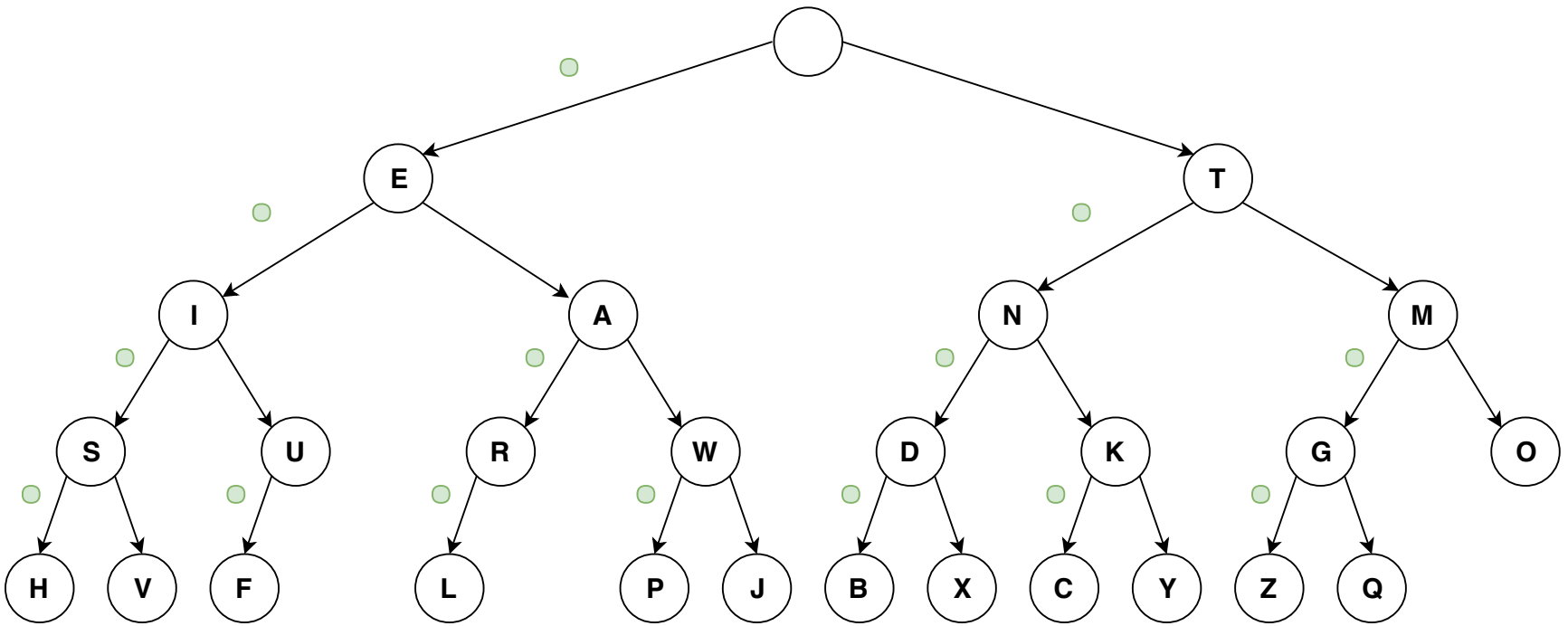
```
node_t* tree_insert(int value, node_t *node)
{
  if (node == NULL) {
    return newNode(value); // vracíme nový uzel
  } else {
    if (value <= node->value) { //vložení do levého
   podstromu
      node->left = tree_insert(value, node->left);
    } else { // vložení do pravého podstromu
      node->right = tree_insert(value, node->right);
    }
    return node; // vracíme vstupní uzel!!!
  }
}
```

Průchod binárním vyhledávacím stromem

- Při hledání prvku konkrétní hodnoty se postupne zanořujeme hlouběji do stromu.
- Může nastat jedna z následujících situací:
  - Aktuální prvek má hledanou hodnotu klíče
    - hledání je ukončeno
  - Hodnota klíče je menší než hodnota aktuálního prvku
    - pokračujeme v hledání v další úrovni levého potomka
  - Hodnota klíče je větší než hodnota aktuálního prvku
    - pokračujeme v hledání v další úrovni pravého potomka
  - Aktuální prvek má hodnotu null
    - hledání je ukončeno, prvek ve stromu není
- Při průchodu stromem postupujeme rekurzivně
  - nejdříve navštěvujeme levé potomky
  - následne pak pravé potomky

Příklad – dekódování morseovky

- Pro dekódování textu zapsaného v Morseově abecedě lze použít následující binární strom



Příklad – dekódování morseovky

```
typedef struct MUzel {
  char znak;
  struct MUzel *tecka, *carka;
} MUzel;

MUzel *novyMUzel(char z, MUzel *t, MUzel *c) {
  MUzel *p = (MUzel*)malloc(sizeof(MUzel));
  p->znak = z; p->tecka = t; p->carka = c;
  return p;
}

MUzel *novyMUzel1(char z) {
  MUzel *p = (MUzel*)malloc(sizeof(MUzel));
  p->znak = z;
  p->tecka = NULL; p->carka = NULL;
  return p;
}
```

Dekódování morseovky – vytvoření stromu

```
MUzel *strom(void) {
  return novyMUzel(' ',
    novyMUzel('E', /* . */
      novyMUzel('I', /* .. */
        novyMUzel('S', /* ... */
          novyMUzel1('H'), /* .... */
          novyMUzel1('V') /* ...- */
        ),
        novyMUzel('U', /* ..- */
          novyMUzel1('F'), /* ..-. */
          NULL /* ..-- */
      )),
    novyMUzel('A', /* .- */ ...)),
  novyMUzel('T', /* - */ ...));
}
```

Dekódování morseovky – dekódovací funkce

```
void dekoduj(char odkud[], char kam[]) {
  MUzel *aktualni = koren;
  int i = 0, j = 0, delka = strlen(odkud);
  while (i<delka) {
    char z = odkud[i];
    if (aktualni!=NULL) {
      if (z=='.') aktualni = aktualni->tecka;
      else if (z=='-') aktualni = aktualni->carka;
      else { kam[j++] = aktualni->znak; aktualni = koren; }
      i++;
    } else {
      kam[j++] = '?';
      while (odkud[i]=='.' || odkud[i]=='-') i++;
      aktualni = koren;
    }
  }
  kam[j] = 0;
}
```

Dekódování morseovky – hlavní program

```
MUzel *koren;
#define MAXDELKA 100

int main(void) {
  koren = strom();
  char morse[MAXDELKA], text[MAXDELKA];
  fgets(morse, MAXDELKA, stdin);
  dekoduj(morse, text); // + uklid strom
  fprintf(stdout, "%s\n", text);
  system("PAUSE");
  return 0;
}
```

Část II

Preprocesor

II. Preprocesor

Preprocesor

Direktivy preprocesoru

- Vždy uvozeny znakem `#`
- `#` na řádku prvním jiným znakem, než jsou odsazovače
- Není příkaz jazyka C – neukončujeme středníkem

- vkládání textu: `#include`
- definice maker: `#define`, `#undef`
- podmíněný překlad: `#if`, `#ifdef`, `#ifndef`, `#else`, `#elif`, `#endif`
- ostatní: `#error`, `#line`

Parametrická makra

```
#define SQUARE(n) (n*n)
#define MAX(x,y) ((x)>(y)?(x):(y))
```

- Formální parametry by se měly vyskytovat v definici makra
- Počet formálních a skutečných parametrů musí souhlasit
- Expanze probíhá textově – závorky zvyšují bezpečnost makra
- **Výhoda:** nepracuje s typy
- **Nevýhoda:** nepracuje s typy

Příklad

```
a = SQR(b)       // a = ((b)*(b))
a = SQR(a+1)     // a = ((a+1)*(a+1))
a = MAX(b,c+1); // a = ((b)>(c+1)?(a):(c+1));
```

Předdefinovaná makra

`__DATE__`	datum překladu mmm dd yyyy
`__FILE__`	jméno zdrojového souboru
`__LINE__`	právě zpracovávaný řádek ve zdrojovém souboru
`__STDC__`	typ (úroveň) překladu (STanDard C)
`__TIME__`	čas překladu hh:mm:ss

Všechna předdefinovaná makra překladače gcc:

```
$ gcc -dM -E - < /dev/null
```

ANSI C doplňky

- Operátor # – předávání textových parametrů makra

```
#define printerror(n)     printf (#n "\n")
```

- Operátor ## – spojování lexikálních elementů

```
#define printvar(n)     printf ("%d\n", var##n)
```

- Příkaz #error – hlášení chyb během zpracování zdrojového kódu preprocesorem

```
#if SIZE < 10
#error Size less then 10
#endif
```

Chybové hlášení se zobrazí na standardní výstup.

Část III

Zadání domácího úkolu

Zadání 8. domácího úkolu (HW08)

Téma: Zpracování strukturovaného textu

- **Motivace:** Analýza textu ve formátu XML
- **Cíl:** Procvičit si práci s abstraktními datovými strukturami, řetězci a dynamickou alokací
- **Zadání:** `https://cw.fel.cvut.cz/wiki/courses/b0b99prpa/hw/hw08`
- **Termín odevzdání:** **5.1.2019, 23:59:59**