Pracovní list 7: Rekurze. Stromy

Co už máme znát

- iterace, rekurze;
- stromové struktury, obecné, pravidelné, uspořádané;
- binární vyhledávací strom;
- vyhledávání a řazení pomocí BVS;
- implementace BVS dynamickou strukturou.

Kontrolní otázky

7.1 Jaké vlastnosti má iterativní řešení problému?

7.2 Jak se liší rekurzivní řešení problému od iterativního?

7.3 Ve kterých úlohách je výhodné využít rekurzivního řešení?

7.4 Jak je definován v teorii grafů strom?

7.5 Co je to uspořádaný strom?

7.6 K čemu lze využít binární uspořádaný strom?

7.7 Jak lze implementovat uzel binárního stromu dynamickou strukturou?

7.8 Jaké druhy průchodů stromem existují?

Příprava na cvičení

Ve cvičení budeme potřebovat překladač jazyka C++, editor pro přípravu zdrojových textů a vybavení příkazového řádku. Pro jednotlivé úlohy jsou k dispozici soubory s daty, případně výsledné soubory v adresáři `/home/rybicka/vyuka/progt/cecko/cviceni/cv07` na serveru akela. Konkrétní jména těchto souborů jsou uvedena u jednotlivých úloh.

Řešené příklady

Příklad 7.1 Definujte rekurzivní výpočet celočíselné mocniny reálného čísla a proveďte jeho implementaci.

Řešení: Máme-li vypočítat celočíselnou mocninu, můžeme to provést postupným násobením daného čísla. Rekurzivní definice může mít následující tvar:

$$C^n = C^{n-1} \cdot C \quad (1)$$
$$C^0 = 1 \quad (2)$$

Příslušnou mocninu zapíšeme jako funkci s dvěma parametry – základem C a požadovanou mocninou n. Tato funkce bude pracovat ve dvou větvích – pro hodnotu $n > 0$ podle prvního řádku definice, pro hodnotu $n = 0$ pak podle druhého řádku. Chybový stav, kdy bude $n < 0$, vyřešíme tím, že funkce bude vracet nulu.

```
256 #include <iostream>
257 using namespace std;
258
259 double Mocnina(double C, unsigned int n){
260   if (n>0) return C * Mocnina(C, n-1); //rekurzivní výpočet
261      else if (n==0) return 1;  //rekurzivní zarážka
262           else return 0;  //ošetření chybného vstupu
263 }
264
265 int main(){
266    double cislo;
267    unsigned int moc;
268    cin >> cislo >> moc;
269    cout << Mocnina(cislo, moc) << endl;
270   return 0;
271 }
```

Příklad 7.2 Implementujte obecný strom (počet následníků v každém uzlu může být libovolný). Datovou složkou uzlu je celé číslo. Naplňte tento strom daty ze standardního vstupu, kde se nachází posloupnost čísel zakončená nulou představující cestu k danému uzlu a za ní následuje číslo vkládané do nového uzlu. Vypište pak celý strom na standardní výstup.

Řešení: K řešení potřebujeme vytvořit obraz uzlu takového stromu. Jde o záznam, jehož složkou je celočíselná datová složka a odkazy na následníky. V tomto případě vyřešíme odkazy jako pole ukazatelů s max. 100 složkami. Alternativou by byl lineární seznam odkazů na následníky, kde nemusíme stanovovat implementační konstantu udávající maximální počet následníků.

S tímto stromem potřebujeme udělat dvě operace: vložení uzlu a průchod stromem. Přidání uzlu představuje nalezení příslušného místa a alokaci struktury, jejíž ukazatel se vloží do odpovídajícího prvku pole uzlu, k němuž bude nový uzel připojen.

```
272 #include <iostream>
273 using namespace std;
274
275 typedef struct Uzel {
```

```
276   int data;
277   unsigned int pocnasled; //aktuální počet následníků
278   Uzel *nasled[100]; //v každém uzlu je 100 možných následníků
279 } Uzel;  //definice uzlu stromu
280
281 typedef Uzel *UkUzel;  //ukazatel na uzel
282 typedef UkUzel Strom;  //strom je reprezentován ukazatelem na kořen
283
284 void Pridej(Strom &U){   //přidání uzlu
285        U = new Uzel;     //U je předáno odkazem, vytvoří se tedy i vazba
286        cin >> (U->data); //data se vloží ze vstupu
287        U->pocnasled = 0; //nový uzel je listem
288 }
289
290 void Najdi(Strom &U){
291   int kam;
292   UkUzel pom;
293   cin >> kam;
294   if (kam==0) Pridej(U); //je dosaženo uzlu
295   else {
296       pom=U->nasled[i]; //jinak sekvenční hledání následníka
297       while ((i<U->pocnasled) and (U->nasled[i]->data!=kam)) i++;
298       Najdi(U->nasled[i]);  //vstup do další hladiny
299   }
300 }
301
302 void Pruchod(Strom U){
303    if (U!=NULL) {
304        cout << U->data << " ";  //výpis daného uzlu
305        for (int i=0; i<U->pocnasled; i++)
306           Pruchod(U->nasled[i]); //výpis všech následníků
307    }
308 }
309
310 int main(){
311   Strom S=NULL; //prázdný strom
312   Najdi(S);
313   Pruchod(S);
314   return 0;
315 }
```

Příklady

Příklad 7.3 Rozšiřte příklad 7.1 tak, aby počítal i záporné celočíselné mocniny.

Příklad 7.4 Implementujte iterativní výpočet *n* členů Fibonacciho posloupnosti a vyzkoušejte řešení pro výpis jejích prvních 100 členů.

Příklad 7.5 Implementujte rekurzivní výpočet *n* členů Fibonacciho posloupnosti a vyzkoušejte řešení pro výpis jejích prvních 15 členů. Porovnejte spotřebovaný čas výpočtu s iterativní metodou.

Příklad 7.6 Napište rekurzivní zadání a pak implementujte vytváření lineárního jednosměrného seznamu ze vstupních celočíselných hodnot v souboru `hodnoty.txt`. Seznam pak opište na standardní výstup.

Příklad 7.7 Předpokládejte, že na standardním vstupu je řada řetězců, na každém řádku jeden. Implementujte rekurzivním způsobem výpis vstupních řetězců v obráceném pořadí. Zajistěte, aby se vypisovaly pouze ty řetězce, které byly načteny ze vstupu. Pro ověření lze na vstup přesměrovat soubor `data.txt`.

Příklad 7.8 Předpokládejte, že na každém řádku souboru `cesty.txt` se nachází jméno diskového souboru s absolutní cestou v unixovém tvaru (cesta začíná lomítkem a mezi adresáři je rovněž oddělovačem lomítko). Načtěte tyto údaje a sestavte z nich v paměti stromovou strukturu adresářů a souborů. Použijte koncept obecného stromu z řešeného příkladu 7.2.

Příklad 7.9 Využijte implementaci předchozího příkladu k této úloze: na standardním vstupu se nacházejí jména adresářů. Vypište obsah zadaného adresáře z údajů, které máte ve stromové struktuře v paměti.

Příklad 7.10 Předpokládejte, že v textovém souboru `hodnoty.txt` se nacházejí celočíselné hodnoty. Uložte je do binárního vyhledávacího stromu. Pak zadávejte ze standardního vstupu hodnoty a na výstup vypisujte informaci o tom, zda se zadaná hodnota nachází ve stromu.

Příklad 7.11 Na standardním vstupu se nachází řada řetězců, na každém řádku jeden. Seřaďte tyto řetězce podle délky pomocí binárního uspořádaného stromu a vypište je na výstup. Pro vstup lze využít soubor `data.txt`.

Příklad 7.12 Doplňte předchozí úlohu o výpis počtu hladin použitého stromu. Porovnejte tento počet s teoretickým počtem hladin u stromu nutných k uložení daného počtu hodnot.

Příklad 7.13 Příklad 7.11 spusťte s tím, že na vstup přesměrujete soubor `data_ne.txt` a zjistíte nyní počet hladin použitého stromu. Jak se liší počet hladin od předchozího případu se souborem `data.txt` a proč? Jaká podstatná vlastnost binárního uspořádaného stromu z toho plyne?

Co máme po cvičení umět

- Rekurzivní zadání a řešení úlohy.
- Vhodnost iterativního nebo rekurzivního řešení problému.
- Princip stromové struktury.
- Binární vyhledávací strom a jeho využití k vyhledávání a řazení.

Kontrolní otázky

7.9 Co musí obsahovat rekurzivní definice problému, aby nedošlo k nekonečné rekurzi?

7.10 Jaký je princip rekurzivní implementace problému?

7.11 V čem spočívá nebezpečí rekurzivního řešení?

7.12 Jak lze implementovat obecnou stromovou strukturu?

7.13 Jaké vlastnosti má binární vyhledávací strom s ohledem na pořadí vkládaných dat?

7.14 Jak se implementuje vyhledání ve stromu?

7.15 Jak se implementují průchody stromem?