Rekurze, vyhledávání a řazení

IZP-cv09

Ing. Jakub Husa

Vysoké Učení Technické v Brně, Fakulta informačních technologií
Božetěchova 1/2. 612 66 Brno - Královo Pole
ihusa@fit.vut.cz

29. listopadu 2023

Rekurze

Rekurze je programovací technika při které funkce volá sebe sama:

```
1  void foo(int n)          //rekurzivni funkce s ukoncujici podminkou
2  {
3      printf("%i\n", n); //funkce vypisuje hodnotu "N"
4      if (n > 0)           //pokud N neni nula (ukoncujici podminka)
5          foo(n-1);        //funcke vola sebe sama pro "N-1"
6      printf("%i\n", n); //funkce vypisuje hodnotu "N"
7  }
```

Rekurzivní volání vždy musí mít nějakou ukončovací podmínku:

- Při každém volání na zásobníku vytváří nová kopie parametrů funkce a nekonečná rekurze tedy vždy havaruje na nedostatek paměti!

```
8  void bar(int n)          //rekurzivni funkce bez ukoncujici podminky
9  {
10     printf("%i\n", n); //funkce vypisuje hodnotu "N"
11     bar(n+1);            //CHYBA -- cca po 65000 volanich rekuze havaruje
12     printf("%i\n", n); //a tento radek se nikdy neprovede
13 }
```

Využití rekuze

Rekurzi používáme pro řešení rekurzivně definovatelných problémů:

- Například – výpočet faktoriálu definovaného rovnicí:

$$0! = 1 \qquad n! = n * (n-1)!$$

```
1  int faktorialRekurzi(int n)
2  {
3      if (n == 0)                          //pokud pocitame faktorial nuly
4          return 1;                        //vysledek je jedna
5      return n * faktorialRekurzi(n-1); //jinak vracime rekuzi
6  }
```

Jednoduché rekurzivní problémy můžeme řešit také pomocí cyklu:

```
7  int faktorialCyklem(int n)
8  {
9      if (n == 0)                          //pokud pocitame faktorial nuly
10         return 1;                        //vysledek je jedna
11     int x = 1;                           //jinak cislo jedna
12     for (int i = 1; i <= n; i++)         //budeme postupne nasobit
13         x = x * i;                       //vsemi hodnotami od 1 do N
14     return x;                            //a vratime vysledny nasobek
15 }
```

Vyzkoušejte si:

- Implementujte funkce fibRek a fibCyk které spočítají hodnotu n-tého členu Fibonacciho posloupnosti definované rovnicí:

$$\text{fib}(0) = 0 \qquad \text{fib}(1) = 1$$
$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$

```
1 int fibRek(int n);
2 int fibCyk(int n);
```

- Funkce fibRek pro výpočet použije rekurzi, fibCyk pro výpočet použije cyklus.
- Ve funkci main načtete celé číslo, obě funkce zavolejte, a vypište výsledek.
- Posloupnost čísel je: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, ...

Například:

- (2) => fibRek(2) = 1
  => fibCyk(2) = 1
- (4) => fibRek(4) = 3
  => fibCyk(4) = 3
- (10) => fibRek(10) = 55
  => fibCyk(10) = 55

Vyhledávání

Sekvenční vyhledávání

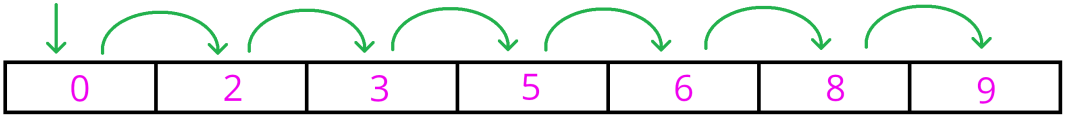
U algoritmů nás zajímá jejich časová (nebo paměťová) asymptotická složitost:

- Udává maximální počet kroků potřebných pro zpracování n položek.
- Více se o složitosti algoritmů budete učit ve třetím semestru, v předmětu IAL.

Například – sekvenční vyhledávání prvku v poli má lineární časovou složitost O(n):

- Algoritmus projde přes všechny prvky pole a porovná je s hledanou hodnotu.
- Prohledat 1 000 000 prvků zabere 1 000 000 kroků.

```
1  int sekvencniVyhledavani(int delka, int pole[], int cislo)
2  {
3      for(int i = 0; i < delka; i++) //prochazime vsemi prvky pole
4          if (cislo == pole[i])      //pokud najdeme hledany prvek
5              return i;              //vratice jeho index
6      return -1;                     //pokud jsme prvek nenasli vracime -1
7  }
```

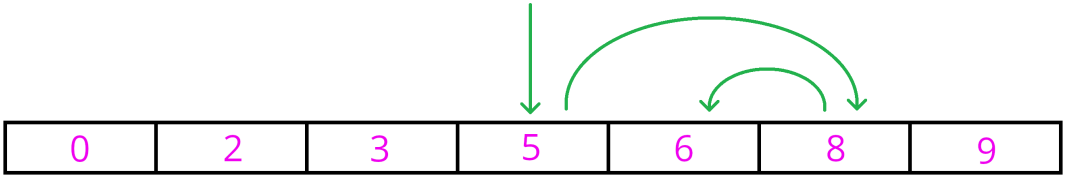


Pokud je pole seřazené můžeme použít algoritmus binárního vyhledávání:

- Binární vyhledávání začíná prostředním prvkem pole.
- Pokud jsme našli hledaný prvek, algoritmus vrátí jeho index.
- Pokud už se pole nedá znovu rozpůlit, tak prvek v poli není.
- Pokud jsme hledaný prvek nenašli, algoritmus pokračuje prostředním prvkem levé nebo pravé poloviny zbytku pole.

Binární vyhledávání má logaritmickou časovou složitost $O(\log(n))$:

- Prohledat 1 000 000 prvků zabere pouze 20 kroků!



Vyzkoušejte si:

- Napište funkci hledej která implementuje binární vyhledávání pomocí rekuze.

```
1  int hledej(int zacatek, int konec, int pole[], int cislo);
```

- Parametry zacatek a konec jsou první a poslední aktuálně prohledávaný index, pole je prohledávané pole a cislo je číslo které se v něm snažíme najít.
- Návratovou hodnotou je index na kterém bylo hledané číslo nalezeno nebo hodnota -1 pokud číslo v poli není.
- Ve funkci main si vytvořte seřazené pole sedmi celých čísel, zavolejte funkci hledej, a vypište její výsledek.

Například:

- ((0, 2, 3, 5, 6, 8, 9), 0) => cislo 0 nalezeno na indexu 0
- ((0, 2, 3, 5, 6, 8, 9), 3) => cislo 3 nalezeno na indexu 2
- ((0, 2, 3, 5, 6, 8, 9), 4) => cislo 4 nenalezeno

Řazení

Řazení

Pro řazení prvků existuje mnoho algoritmů s různými vlastnostmi:

- Pomalé řadící algoritmy (Bubble-sort, Select-sort, Insert-sort, ...) používají dvojitý cyklus, mají kvadratickou časovou složitost $O(n^2)$, a seřadit 1000 prvků jim zabere 1 000 000 kroků.
- Rychlé řadící algoritmy (Quick-sort, Merge-sort, Heap-sort, ...) používají rekurzi a cyklus, mají linearitmickou časovou složitost $O(n * log(n))$, a seřadit 1000 prvků jim zabere jen 10 000 kroků

Například – implementace algoritmu bublinkového řazení celých čísel:

```
1  void bubbleSort(int delka, int pole[])    //bublinkove razeni
2  {
3      for(int i = 0; i < delka; i++)        //pro kazdy prvek
4          for(int j = 0; j < delka-1; j++) //projdi vsecny prvky
5              if(pole[j] > pole[j+1])       //pokud jsou dva sousedni
6              {                             //prvky ve spatnem poradi
7                  int temp = pole[j];       //vymen jejich hodnoty
8                  pole[j] = pole[j+1];
9                  pole[j+1] = temp;
10             }
11 }
```

Vyzkoušejte si:

- Implementujte následující funkce pro práci s osobami:

```
1  int porovnej(osoba A, osoba B);
2  void vymen(osoba* A, osoba* B);
3  int nejmensi(int delka, osoba pole[]);
4  void serad(int delka, osoba pole[]);
```

- porovnej porovná jména dvou osob a pokud jsou stejná tak porovná jejich věk. Pokud je první osoba větší tak funkce vrátí kladné číslo, pokud je první osoba menší tak vrátí záporné číslo, pokud jsou obě osoby stejné tak vrátí nulu.
- vymen spolu dvě osoby vzájemně vymění.
- nejmensi vrátí index osoby s nejmenší hodnotou (dle funkce porovnej).
- serad pole osob seřadí, například algoritmem Selection-sort nebo Bubble-sort.

```
5  (Bob 23) (Cecil 23) (Bob 22) (Alice 24) (Daniela 21)
6  Ruzna jmena     (0,1) OK
7  Ruzny vek       (0,2) OK
8  (Daniela 21) (Cecil 23) (Bob 22) (Alice 24) (Bob 23)
9  Vymena   osob   (0,4) OK
10 Nejmensi osoba (3)   OK
11 (Alice 24) (Bob 22) (Bob 23) (Cecil 23) (Daniela 21)
```

Na procvičení

```c
typedef struct Sosoba //deklarujeme datovy typ pro strukturu
{                     //jmenem "Sosoba" se dvema polozkami
    char jmeno[10];   //pole znaku jmenem "jmeno"
    int vek;          //cele cislo jmenem "vek"
} osoba;              //jmeno tohoto typu je "osoba"

void vypis(int delka, osoba pole[]);    //deklarace funkce "vypis"
int porovnej(osoba A, osoba B);         //deklarace funkce "porovnej"
void vymen(osoba* A, osoba* B);         //deklarace funkce "vymen"
int nejmensi(int delka, osoba pole[]); //deklarace funkce "nejmensi"
void serad(int delka, osoba pole[]);    //deklarace funkce "serad"

void vypis(int delka, osoba pole[])     //definice funkce "vypis"
{
    for(int i = 0; i < delka; i++)      //pro vsechny prvky pole
    {
        printf("(%s %i) ", pole[i].jmeno, pole[i].vek); //vypis
    }                   //jmeno a vek vsech osob, oddelene mezerami
    printf("\n");       //vypis konec radku
}
```

Na procvičení

```
1  int main()
2  {
3      osoba pole[5] = {{"Bob", 23},      //osoba 0
4                       {"Cecil", 23},    //osoba 1
5                       {"Bob", 22},      //osoba 2
6                       {"Alice", 24},    //osoba 3
7                       {"Daniela",21}}; //osoba 4
8
9      vypis(5, pole);
10     if (porovnej(pole[0], pole[1])<0) printf("Ruzna jmena     (0,1) OK\n");
11     if (porovnej(pole[0], pole[2])>0) printf("Ruzny vek       (0,2) OK\n");
12
13     vymen(&pole[0], &pole[4]);
14     vypis(5, pole);
15     if (porovnej(pole[0], pole[4])>0) printf("Vymena    osob   (0,4) OK\n");
16     if (nejmensi(5, pole) == 3)       printf("Nejmensi osoba (3)    OK\n");
17
18     serad(5, pole);
19     vypis(5, pole);
20     return 0;
21 }
```