Seminár z Algoritmizácie a Programovania

Okruhy k záverečnému testu.

Obsah

1. Čo sú prvočísla?

Prvočíslo je číslo, ktoré je **väčšie ako 1** a ktorého **jedinými deliteľmi je to samotné číslo a jednotka** (1 a 0 sem nepatria). Všetky prirodzené čísla ktoré nie sú prvočíslami sa nazývajú **zložené čísla.** Skúmaním sa zaoberá **teória čísel.**

Mersenovo číslo

Najväčšie prvočíslo má 23 miliónov znakov - **Marsennovo číslo** – vypočítané pomocou vzorca (2^n)-1 ⇒ toto sa ale nedá použiť na výpočet všetkých prvočísel. Prvočísla, ktoré sa dajú napísať v **tvare** 2^{n-1} sa nazývajú **Mersennove prvočísla**. Môžeme si všimnúť, že aj zatiaľ najväčšie známe prvočíslo je Mersennovo prvočíslo. Ak 2^{n-1} je prvočíslo, tak **aj** n **je prvočíslo**.

Femratove čísla

Čísla v tvare 2^{2n+1} sa nazývajú Fermatove čísla.

Faktorizácia

Faktorizácia je rozklad prirodzených čísel na prvočísla (4=2*2; 6=2*3; 7=1*7)... V súčasnosti sa ale ešte nenašiel algoritmus na faktorizáciu veľkých čísel.

a. Ako sa prvočísla používajú v informatike?

Každé prirodzené číslo väčšie ako 1 sa dá napísať ako **súčin prvočísel.** Tento súčin voláme **prvočíselný rozklad.**

Kryptografia s verejným kľúčom, ktorú zaraďujeme do kryptografie s asymetrickým kľúčom, konkrétne tzv. **RSA kryptosystémom**. Práve tu sa ukázali prvočísla ako veľmi dôležité, pretože RSA kryptosystém na šifrovanie využíva **rozklad veľkých celých čísel na súčin prvočísel**, čo je výpočtovo náročný proces.

Multiplying two numbers, even if very large, is perhaps tedious but a straightforward task. **Finding prime factorization,** on the other hand, **is extremely hard**, and that is precisely what the **RSA** system takes advantage of.

b. Algoritmus na vyhľadanie n-tého prvočísla.

Ak n je zložené číslo a teda má deliteľov, tak má zaručene aj prvočíselných deliteľov - nemusíme teda skúšať všetky čísla ≤√n, ale len všetky prvočísla ≤√n, ktorých je π(√n).

```
void najdi_prvocisla (int n) {
    for (int i = 2; i <= n; i++) {
        bool ok = true;
        for (int j = 0; j < prvocisla.size() && prvocisla[j] * prvocisla[j]
<= i; j++) {
            if (i % prvocisla[j] == 0) {
                ok = false;
                break;
            }
        }
        if (ok) {
            prvocisla.push_back(i);
        }
    }
}
```

c. Algoritmus na vyhľadanie n prvočísel.

Eratostenovo sito je jednoduchý algoritmus pre nájdenie všetkých prvočísel menších ako zadaná horná hranica. Tento algoritmus ide na problém opačne - **každým prvočíslom prejdeme všetky jeho dostatočne veľké násobky a označíme ich ako zložené.**

```
vector<int> prvocisla;
void Eratosten (int n) {
  vector <bool> zlozene (n+1, false);
  zlozene[0] = zlozene[1] = true;
  for (long long i=2; i <= n; i++) {
    if (zlozene[i]) {
      continue;
    }
    prvocisla.push_back(i);
    for (long long k = i*i; k<=n; k += i) {}
  }
}
```

d. Algoritmus na vyhľadanie prvočísel v interval.

to isté?

2. Práca s veľkými číslami

Polia v jazyku C umožňujú pomerne efektívne reprezentovať dlhé čísla s veľkým počtom cifier. Zabudované jednoduché typy majú **obmedzený rozsah hodnôt** s ktorým dokážu pracovať, pričom číslo väčšie ako tento rozsah nie je možné v premennej uložiť. Typ **unsigned int** (32 bitov) dokáže reprezentovať najviac 4 294 967 295. Typ **unsigned long** (64 bitov) najviac 18 446 744 073 709 551 615.

Najjednoduchšia reprezentácia dlhých čísel **je v poli znakov (typ char) konštantnej dĺžky**. V závislosti od typu dlhých čísel, ktoré chceme reprezentovať, môžu byť prvky poľa ľubovoľného dátového typu, ktorý dokáže reprezentovať všetky cifry čísla: v prípade desiatkových čísel dokáže typ char pohodlne reprezentovať hodnoty 0, 1, ..., 9. Konštantná dĺžka nám zabezpečí, že pri operáciách nemusíme uvažovať nad správnou dĺžkou, ktorá sa môže operáciami priebežne zväčšovať (prenosom do vyšších rádov).

	0	1	2	3	4	5	...	99
reťazec	'1'	'2'	'3'	'4'	'\0'	??	??	??
ASCII kód	49	50	51	52	0	??	??	??
Dlhé číslo	4	3	2	1	0	0	0	0

Rozdiel je ten, že v prípade dlhých čísel reprezentujeme **v poli** cifry postupne od najnižších rádov (4,3,2,1), pričom **v prípade reťazcov** (ľudsky čitateľnej formy čísla) sú cifry uvedené od najvyšších rádov (1,2,3,4). A tiež všetky cifry v poli pre dlhé číslo musia mať inicializovanú hodnotu, naopak v prípade reťazcu môžu byť hodnoty znakov za ukončovacím znakom '\0' ľubovoľné.

```
#define MAX_DLZKA_CISLA 1000            // prevod medzi dlhym cislom a polom
char *dlhecislo(const char *str) {
  int i, n = strlen(str);
  if (n > MAX_DLZKA_CISLA)
   return NULL; // prilis dlhy retazec
  char *dlhe = calloc(MAX_DLZKA_CISLA, 1);
  for (i = 0; i < n; i++)
    dlhe[i] = str[n - i - 1] - '0';
  return dlhe;
}

char *retazec(const char *dlhe) {
  char str[MAX_DLZKA_CISLA];
  int i, j = 0;
  for (i = MAX_DLZKA_CISLA - 1; i > 0; i--)
    if (dlhe[i] > 0)
      break;
  while (i >= 0)
    str[j++] = dlhe[i--] + '0';
  str[j] = 0;
  return strdup(str);
}
```

3. Triediace algoritmy - princíp algoritmov, zdrojový kód

a. Prehľad triediacich algoritmov - Algoritmy triediace v čase O(n2)

i. Bubblesort

klasika

ii. Insertsort

2 polia: Jedno vstupné jedno výstupné, nachádzame minima vo vstupnom, zapisujeme do konečného a dávame preč zo vstupného:

```
void insertionSort(int array[], int size) {
  for (int step = 1; step < size; step++) {
    int key = array[step];
    int j = step - 1;
    while (key < array[j] && j >= 0) {
      array[j + 1] = array[j];
      --j;
    }
    array[j + 1] = key;
  }
}
```

iii. Selectsort

```
void selectionSort(int array[], int size) {
  for (int step = 0; step < size - 1; step++) {
    int min_idx = step;
    for (int i = step + 1; i < size; i++) {
      if (array[i] < array[min_idx])
        min_idx = i;
    }
    swap(&array[min_idx], &array[step]);
  }
}
```

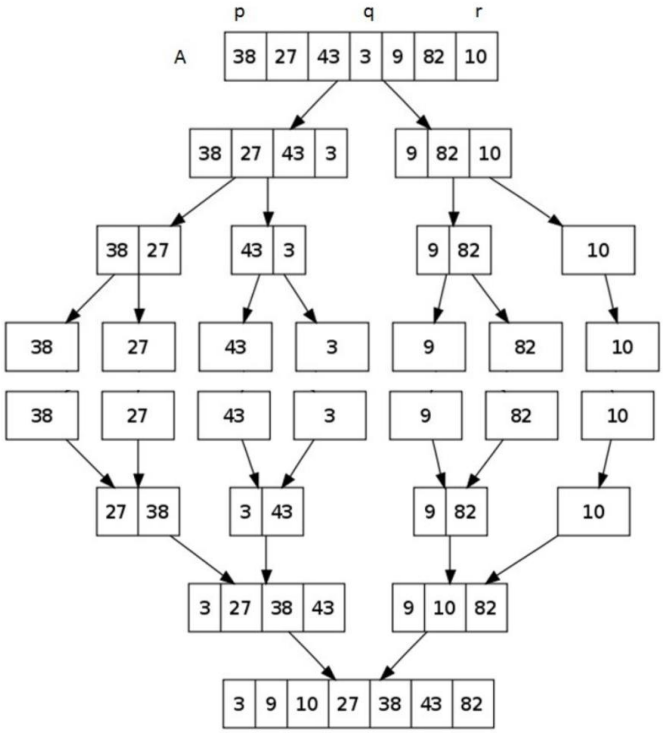
b. Algoritmy triediace v čase O(n log n)

i. **Quicksort**

https://www.programiz.com/dsa/quick-sort

ii. **Mergesort -** typický rekurzívny

https://www.geeksforgeeks.org/merge-sort/
https://www.programiz.com/dsa/merge-sort



iii. **Heapsort**

https://www.geeksforgeeks.org/heap-sort/

c. Algoritmy triediace v čase O(n)

Z hľadiska časovej zložitosti sú najvýkonnejšími algoritmy tie, ktoré **neporovnávajú jednotlivé hodnoty** prvkov, ale fungujú na inom princípe (zložitosť O (n)).

i. **Countingsort**

??? Počíta početnosť prvkov a potom nejak súčet a ???

ii. **Radixsort**

Porovnáva od poslednej číslice (10**1**, 50**9**, 08**7**...) a potom ďalej..

iii. **Bucketsort**

Rozdelí čísla najskôr podľa kategórií (0-5), (5-10)...

4. Vyhľadávacie algoritmy princíp algoritmov, zdrojový kód

http://www2.fiit.stuba.sk/~pospichal/soltis/uvod.htm

a. Algoritmy pre neusporiadané polia

i. Sekvenčné vyhľadávanie

b. Algoritmy pre usporiadané polia

i. Sekvenčné vyhľadávanie (s podmienkou) – prvok za prvkom

ii. Binárne vyhľadávanie

```
int najdi (int* pole, int velkost, int hladaj) {
  int pravy_ind = velkost - 1;
  int lavy_ind = 0;

  while (lavy_ind <= pravy_ind) {
    int stred = ((lavy_ind + pravy_ind) / 2);
    if (pole[stred] < hladaj) {
      lavy_ind = stred+1;
    }
    else if (pole[stred] > hladaj) {
      pravy_ind = stred-1;
    }
    else
      return stred;
  }
}
```

c. Algoritmy pre vyhľadávanie vzoru v reťazci

palindromy??

5. Viacrozmerné polia

```
char **pole;

void alokuj (int *vyska, int *sirka) {
  // alokovanie miesta pre pointre na riadky
  pole = (char **) malloc (*vyska *sizeof (char *));
  // alokovanie miesta pre konkretne riadky
  for (i=0; i < *vyska; i++){
    pole[i] = (char *) malloc (*sirka+1 *sizeof (char)); // +1 pre nulovy
znak
  }
}
```

6. Bitové operácie

https://www.hackerearth.com/practice/basic-programming/bit-manipulation/basics-of-bit-manipulation/tutorial/

X	Y	X&Y	X|Y	X^Y	~(X)
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Pre účely manipulácie s bytmi poskytuje jazyk C 6 operátorov:

```
& - bitový súčin (AND)

| - bitový súčet (OR)

^ - bitový exkluzívny súčet (XOR)

<< - posun doľava

>> - posun doprava

~ - jednotkový doplnok - negácia bit po bite - unárny operátor
```

Bitový súčin : i-ty bit výsledku bitového súčinu:

```
x & y
```

bude *1*, pokiaľ *i-ty* bit *x* a *i-ty* bit *y* budú *1*, ináč bude *0*. Teda jednotlivé bity výsledku budú záležať na jednotlivých bitoch operandov. **Príklad:**

```
#define je_parne(x) (1 & (unsigned)(x))
```

bitový súčin sa často používa na vymaskovanie (nastavenie na nulu) určitých bitov,
napr. ak chceme premennú typu int previesť na ASCII znak, teda využiť len najnižších 7 bitov:

```
c = c & 0x7F;  /* 0x7F je 0000 0000 1111 1111 */
```

Bitový súčet : i-ty bit výsledku bitového súčtu:

```
x | y
```

bude ***1***, pokiaľ ***i-ty*** bit ***x*** alebo ***i-ty*** bit ***y*** bude *1*, ak budú obidva nulové, bude výsledok ***0***. bitový súčet sa často používa na nastavenie určitých bitov na ***1***, pričom sa ostatné bity nechajú nedotknuté.

Príklad: nasledujúce makro môže byť použité na zmenu veľkých písmen na malé:

```
#define na_male( c ) (c | 0x20) /* 0x20 je 0010 0000 binárne */
```

Bitový exkluzívny súčet : i-ty bit výsledku bitového XOR:

```
x ^ y
```

bude ***1***, pokiaľ ***i-ty*** bit ***x*** sa **nerovná** ***i-temu*** bitu ***y***, ak sú **obidva nulové**, alebo **obidva jednotkové** bude výsledok ***0***. Táto operácia sa dá použiť k porovnaniu dvoch celých čísiel:

```
if (x ^ y)

/* čísla sú rozdielne */
```

Operácia bitového posunu doľava

```
x << n
```

Posunie bity v ***x*** doľava o ***n*** pozícií. Pri tomto posune sa zľava bity **strácajú** - sú vytlačované - a sprava sú doplňované ***0***. Bitový posun doľava sa občas používa na rýchle násobenie dvomi, respektívne mocninou dvoch. **Napr. Príkaz:**

```
x = x << 1;
```

vynásobí ***x*** dvomi, alebo príkaz:

```
x <<= 3;
```

vynásobí ***x*** ôsmimi ($8 = 2^3$)

Negácia bit po bite : na túto akciu sa tiež často používa názov jednotkový doplnok. Príkaz:

```
~x
```

Prevráti nulové bity na jednotkové a naopak. Tento operátor sa používa napr. v situáciách, keď sa chceme vyhnúť počítačovo závislej dĺžke celého čísla. **Napríklad príkaz:**

```
unsigned int x;

x &= 0xFFF0;
```

nastaví na ***nulu*** najnižšie 4 bity ***x***. Bude ale pracovať správne len na počítačoch, kde platí: sizeof(int) == 2.

7. Lineárny zoznam

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

// deklaracia struktury kniha a jej parametrov
typedef struct kniha {
  int poradie_zaznamu;
  char signatura[100];
  long long isbn; // long nestacil a int uz vobec
  // mohol by byt string teoreticky ak vieme ze isbn ma 13 znakov
  char nazov[100];
  char autori[100];
  int datum;
  int preukaz;
  struct kniha* dalsia;
} KNIHA;

// funkcie, ktore sa budu pouzivat
// vsetky, v ktorych sa moze zmenit nieco o zaznamoch (nacitanie novych, mazanie, pridavanie...) musia vracat ukazovatel na
// realne aktualne funkcny prvy zaznam - head, aby dalsie funkcie pracovali s up-to-date udajmi
KNIHA* nacitaj(KNIHA* head); // nacitanie
int skontroluj(KNIHA* head); // kontrola, ci je nieco nacitane
void vypis(KNIHA* head); // vypis zoznamu
KNIHA* uvolni(KNIHA* head); // uvolnenie pamate
KNIHA* pridaj(KNIHA* head); // pridanie zaznamu
void hladaj (KNIHA* head); // hladanie zaznamov
```

```c
// main funkcia
int main() {
  KNIHA* head = NULL; // head node zoznamu knih
  char pokyn, enter;  // vstup od pouzivatela

  // cyklus na nacitavanie prikazov z klavesnice od pouzivatela
  do {
    // nacitaj prikaz
    scanf("%c", &pokyn);

    // ak n => nacitaj
    if (pokyn == 'n') {
      head = nacitaj(head); // funkcia vrati aktualizovany zaciatok
    }
    // ak v => vypis
    if (pokyn == 'v') {
      vypis(head);
    }
    // ak p => pridaj
    if (pokyn == 'p') {
      head = pridaj(head);
    }
    // ak k => koniec
    if (pokyn == 'k') {
      head = uvolni(head);
    }
    if (pokyn == 'h') {
      hladaj (head);
    }

  } while (pokyn != 'k');

  return 0;
}
```

```c
void hladaj (KNIHA* head) {
  KNIHA* prechadzaj;
  int poradie=0, id;

  scanf ("%d", &id);

  prechadzaj = head;

  while (prechadzaj != NULL) {

    // ak sa preukaz v zazname zhoduje s hladanym
    if (prechadzaj->preukaz == id) {
      poradie++;
      // vypis
      printf("%d.\n", poradie);
      printf("SIGNATURA: %s\n", head->signatura);
      printf("ISBN: %lld\n", head->isbn);
      printf("NAZOV: %s", head->nazov);
      printf("AUTORI: %s", head->autori);
      printf("DATUM: %d\n", head->datum);
      printf("PREUKAZ: %d\n", head->preukaz);
    }

    prechadzaj = prechadzaj->dalsia;
  }

  if (poradie == 0) {
    printf ("Pre dane cislo preukazu neevidujeme vypozicky\n");
  }

}
```

```c
// tato funkcia nacitava zaznamy o knihach zo subora kniznica.txt
// je typu KNIHA*, pretoze vracia ukazovatel na prvy zaznam, ktory tiez dostava ako argument
KNIHA* nacitaj(KNIHA* head) {

  KNIHA* predchadzajuca = NULL; // ukazovatel na predchadzajucu knihu - na posuvanie zaznamov
  FILE* subor = fopen("kniznica.txt", "r"); // subor, z ktoreho su citane zaznamy
  char pokracuj[4], enter; // pokracuj sluzi na nacitanie --- zo subora, a enter na citanie entrov \n
  int pocet_knih = 0; // sluzi na cislovanie zaznamv

  if (subor == NULL) {
    printf("ZAZNAMY NEBOLI NACITANE\n"); // ak sa subor nepodarilo otvorit
    return head;
  }

  if (skontroluj(head)) {
    head = uvolni(head); // ak uz boli nacitane nejake filmy uvolni pamat
  }

  // kym su v subore vstupy na precitanie (---)
  while (fscanf(subor, "%s", pokracuj) > 0) {
    fscanf(subor, "%c", &enter); // nacitanie \n ktory nasleduje

    // vytvorenie samostatneho nespojeneho uzlu pre knihu
    KNIHA* temp = malloc(sizeof(struct kniha));
    assert(temp != NULL); // okrem chybnej alokacie, -> exit
    temp->dalsia = NULL; // neukazuje na ziadnu dalsiu knihu
    // (je dolezite aby posledny zaznam ukazoval na NULL lebo inak vypis padne)

    pocet_knih++; // kazdou dalsou vytvorenou knihou sa zvysi pocet knih

    // nacitanie jednotlivych hodnot - informacie idu v stanovenom poradi
    temp->poradie_zaznamu = pocet_knih; // pocet knih sme si vypocitali sami
    fscanf(subor, "%s%c", temp->signatura, &enter); // treba nacitat aj enter lebo by ho potom bral dalsi scanf
    fscanf(subor, "%lld%c", &temp->isbn, &enter); // a myslelo by si to ze to ma ovela viac vstupov a neslo by to
    fgets(temp->nazov, 100, subor);
    fgets(temp->autori, 100, subor);
    fscanf(subor, "%d%c", &temp->datum, &enter);
    fscanf(subor, "%d%c", &temp->preukaz, &enter);
```

```
    // spojenie do zoznamu
    if (head == NULL) { // ak nebolo este nic nacitane a toto je prva kniha
      head = temp; // to co sme nacitali ako prve bude head
      predchadzajuca = temp; // a zatial aj predchadzajuca, vyuzivat sa bude na
svoj ucel az pri dalsich zaznamoch
    }
    else { // ak uz zoznam existuje pripojime knihu na koniec zoznamu
      predchadzajuca->dalsia = temp; // v pripade druheho zaznamu teda head
ukazuje na prave nacitanu knihu
      predchadzajuca = temp; // a nacitana kniha sa akoby posunie na miesto
predchadzanucej,
      // bude sa k nej pripajat (ak existuje) dalsi temp
    }
  }

  printf("NACITALO SA %d ZAZNAMOV\n", pocet_knih); // podla zadania vypiseme
kolko knih sa nacitalo
  fclose(subor); // zavrieme subor
  return head; // funkcia vrati ukazovatel na prvy zaznam
}

// kontroluje ci uz bli nacitane nejake zaznamy a vracia podla toho bud 0 alebo 1
// a.k.a. snaha o dekompoziciu programu
int skontroluj(KNIHA* head) {
  if (head == NULL) {
    return 0; // neboli nacitane filmy
  }
  else {
    return 1; // boli nacitane filmy
  }
}
```

```c
// rekurzivna funkcia na vypis zaznamov
void vypis(KNIHA* head) {

  if (!(skontroluj(head))) {
    return; // ak nie je co vypisat
  }

  // vypis dat
  printf("%d.\n", head->poradie_zaznamu);
  printf("SIGNATURA: %s\n", head->signatura);
  printf("ISBN: %lld\n", head->isbn);
  printf("NAZOV: %s", head->nazov);
  printf("AUTORI: %s", head->autori);
  printf("DATUM: %d\n", head->datum);
  printf("PREUKAZ: %d\n", head->preukaz);

  // ak nasleduje dallsia kniha
  if (head->dalsia != NULL) {
    vypis(head->dalsia); // sprav vyis aj s dalsou knihou
  }
  // volba nazvat parameter tejto funkcie head moze byt mozno trochu matuca
  // kedze nie vzdy naozaj predstavuje prvy zaznam, nakolko sa funkcia posuva
zoznamom

  return;
}

// uvolnenie alokovanej pamate pre zaznamy knih
KNIHA* uvolni(KNIHA* head) {
  KNIHA* temp; // potrebujeme temp aby sme sa vedeli posuvat po zaznamoch

  while (head != NULL) {
    temp = head;
    head = head->dalsia;
    free(temp); // postupne od zaciatku uvolnime vsetky uzly
  }
  head = NULL;
  return head;
  // tato funkcia by nemusela vraciat nic ak by bola volana len pri ukoncovani
programu,
  // ale kedze treba uvolnit pamat aj ak pri nacitani uz zaznamy existuju
  // potrebujeme poznat head
}
```

```c
// pridava individualne zaznamy (zo vstupu od pouzivatela) na urcene miesto C1
KNIHA* pridaj(KNIHA* head) {
  int c1; // pozicia kam sa ma zaznam pridat
  int prazdny = 0; // urcuje ci pridavane novy zaznam do prazdneho zoznamu alebo pridavame do uz existujuceho
  int najdeny = 0; // urcuje, ci C1 reprezentuje existujucu poziciu v zozname
  char enter; // pre nacitanie \n
  KNIHA* prechadzanie = NULL; // na prechadzanie cez zaznamy
  KNIHA* posledna = NULL; // posledny existujuci zaznam v zozname

  scanf("%d", &c1);
  assert(c1 > 0); // ma byt vacsie ako 0 (lebo zaznamy pocitame od prveho)

  if (!(skontroluj(head))) {
    prazdny = 1; // ak nie je nic nacitane
  }

  // na najdenie uzlu s poradim == C1 a uzlu pred nim,
  // alebo na najdenie celkovo posledneho uzlu v zozname v pripade ze pozicia C1 neexistuje
  if (prazdny == 0) { // ak mame nacitane polozky
    prechadzanie = head; // prejdeme zoznam od zaciatku
    // kym nenajdeme zelanu poziciu alebo kym neprideme na koniec zoznamu
    while (prechadzanie != NULL) {
      // v pripade ze prejdeme cely zoznam a nenajdeme zhodu, posledna drzi adresu posledneho ne-NULL-oveho zaznamu
      // ak zhodu najdeme, posledna ukazuje na uzol pred tym, kde je najdena zhoda
      if (prechadzanie->poradie_zaznamu != c1) {
        posledna = prechadzanie;
      }
      else { // ak najdeme zelany zaznam
        najdeny = 1;
        break;
      }
      prechadzanie = prechadzanie->dalsia; // posuvanie sa zoznamom
    }
  }
```

```
// novy uzol
  KNIHA* nova = malloc(sizeof(struct kniha));
  assert(nova != NULL);

  // nacitanie jednotlivych hodnot
  scanf("%s%c", nova->signatura, &enter);
  scanf("%lld%c", &nova->isbn, &enter);
  fgets(nova->nazov, 100, stdin); // cely nazov aj s medzerami
  fgets(nova->autori, 100, stdin);
  scanf("%d%c", &nova->datum, &enter);
  scanf("%d%c", &nova->preukaz, &enter);

  // MOZNOST A - ak je to prvy a jediny zaznam
  if (prazdny == 1) {
    nova->poradie_zaznamu = 1;
    nova->dalsia = NULL;
    return head = nova;
  }

  // MOZNOST B - ak ideme pridat na koniec zoznamu
  else if (najdeny == 0) {
    nova->poradie_zaznamu = (posledna->poradie_zaznamu) + 1; // v poradi
nasleduje za poslednou knihou
    nova->dalsia = NULL; // dalsi zaznam neexistuje
    posledna->dalsia = nova; // posledna ukazuje na novu
    return head;
  }

  // MOZNOST C - ideme nahradit existujuci uzol
  // ak sme C1 nasli prechadzanie  dzri hodnotu zaznamu s poradim == C1 a
posledna toho zaznamu pred nim
  else {
    nova->poradie_zaznamu = c1; // poradie noveho zaznamu

    // treba vsetkym dalsim kniham posunut poradie
    KNIHA* temp = prechadzanie;
    while (temp != NULL) {
      temp->poradie_zaznamu += 1;
      temp = temp->dalsia;
    }
```

```
// AK NAHRADZAME POZICIU UZLU HEAD
    if (prechadzanie == head) {
      nova->dalsia = head; // ukazuje na tu, co mala poradie C1
      head = nova;
    }
    // AK NAHRADZAME KNIHU NIEKDE DALEJ V ZOZNAME
    else {
      nova->dalsia = posledna->dalsia; // nova ukazuje na tu, co mala poradie C1
      posledna->dalsia = nova; // a nahradi jej miesto
    }


    return head; // vrati ukazovatel na prvy zaznam
  }
}
```