

Dátové štruktúry a algoritmy



Aula magna
streda 10:00

letný semester
2020/2021

Dátové štruktúry a algoritmy

Organizácia predmetu

17. 2. 2021

letný semester
2020/2021

Ing. Lukáš Kohútka, PhD.

- Absolvoval Bc. a Ing. na FIIT
- Získal PhD. na FEI v roku 2018
 - Nové architektúry pre zorad'ovanie dát, správu úloh a pamäte
- Celkovo cca. 9 rokov softvérový a hardvérový inžinier
- Výučba predmetov na FIIT/FEI od roku 2014
- Vedúci 21 BP a 11 DP + 2 tímové projekty

Ako sa pracuje na tomto predmete?

- **Prednášky (streda 10:00):**
zapájať sa, pýtať sa, počúvať, robiť si poznámky ...
- **Cvičenia:**
3 zadania
priebežný progres
konzultácie k zadaniam
odovzdanie zadaní
- **Doma:**
vypracovanie veľkých zadaní

Vyučující

- Prednášky (streda 10:00):
Lukáš Kohútka
- Cvičiaci:
Dominika Dolhá
František Horvát
Lukáš Kohútka
Peter Lehoczský
Marián Potočný
Martin Sabo

Podmienky absolvovania

- Môžete získať celkovo 100 bodov
- Priebežne riešené zadania (max. 40 bodov):
 - na cvičení a doma sa budú riešiť 3 zadania
 - prvé za max. 15 bodov (min. 6)
 - druhé za max. 10 bodov (min. 4)
 - tretie za max. 15 bodov (min. 6)
- Podmienky udelenia zápočtu:
 - žiadna neospravedlnená absencia z cvičení
 - celkovo minimálne 20 bodov zo zadaní
 - minimálny počet bodov za každé zadanie (6, 4, 6)
- Záverečná skúška (max. 60 bodov)

Plagiátorstvo

- Všetko, čo sa predkladá na hodnotenie, **musí byť vlastná samostatná práca študenta** alebo musí byť označené ako prevzaté.
- Samozrejme, **body možno získať len za vlastnú prácu**. Opisovanie sa netoleruje.
- Pokiaľ sa pokúšate absolvovať tento predmet nie vlastnou prácou, najmä, ak neupozorníte cvičiaceho, že odovzdané riešenie (alebo jeho časť) je prebratá z iného zdroja, kvalifikujete sa na FX.

Čo chcem, aby ste si odniesli z tohto predmetu

- Prehľad nástrojov (algoritmov a dátových štruktúr) pre riešenie rozličných problémov
- Porozumieť vlastnostiam týchto nástrojov, najmä:
 - časovej efektívnosti (výpočtovej zložitosti)
 - pamäťovej efektívnosti (priestorovej zložitosti)
- Schopnosť aplikovať a prispôbiť-upraviť tieto nástroje pre špecifické problémy (za účelom dosiahnutia čo najlepšej efektívnosti)
- Otvorenosť pre ďalšie štúdium nových algoritmov a použitie efektívnych algoritmov a dátových štruktúr vo vašej ďalšej práci

Dôležité termíny!

- Odovzdanie zadaní - najneskorší možný termín:
 - Zadanie 1 - deň pred 4. cvičením (do 23:59)
 - Zadanie 2 - deň pred 8. cvičením (do 23:59)
 - Zadanie 3 - deň pred 12. cvičením (do 23:59)
 - **Termíny sa neposúvajú!**
(pokúste sa odovzdať vždy o týždeň skôr!)

- Otázky?

Dátové štruktúry a algoritmy

Úvod do problematiky

17. 2. 2021

letný semester
2020/2021

Čo je to algoritmus?

- Na počiatku bolo slovo ...
- Ľudia mali hovorený jazyk, ale prečo vzniklo písmo?
- V starovekej Mezopotámii s miliónom obyvateľov potrebovali registrovať majetok, daňové povinnosti a ich splňanie / nesplňanie
- Informácie zapamätať, vyhľadávať a aktualizovať
- Základným prvkom písma je **abeceda**
 - každá konečná neprázdna množina symbolov

Čísla ako postupnosti symbolov

- V Mezopotámii to bola kombinácia čísel 10 a 6.
- Základné jednotky: 1, 10, 60, 600, 3600, ...
- Prečo tento systém neprežil?
- Prečo používame desiatkovú sústavu?
- Čo očakávame od zápisu čísla?
 - Rýchlo z neho rozumieme, čo tým zapísaným číslom myslíme
 - Efektívne aritmetické operácie
- Efektívne sčítovanie:
 - spojiť karty (symboly) dvoch čísel a potom minimalizácia „kariet“
 - použiť minimálny počet symbolov ... Rímske čísla to mali pôvodne tiež, až neskôr sa začali dávať menšie pred väčšie (napr. IX=9).
- Násobenie ťažko realizovateľné...

Po písme prišla matematika

- Písmo: vedeli sme vytvárať a zapisovať poznatky
- Hľadali sme objektivitu: jazyk, v ktorom, všetko čo napíšeme, sa dá jednoznačne interpretovať...
- A tiež chceme vytvoriť jazyk, v ktorom je každá argumentácia verifikovateľná ...
- **Úlohou matematiky je automatizovať**

Vymyslieť Pytagorovu vetu bolo intelektuálne náročné, ale keď stavební robotníci potrebovali 90° uhly, tak napli šnúru so stranami v pomere 3:4:5 a vedeli, že majú pravý uhol ...

Úlohou matematiky je automatizovať

- Problém je množina inštancií (prípádov) problému ... nekonečná množina.
- Napr. Problém sústavy lineárnych rovníc: existuje nekonečne veľa sústav lineárnych rovníc
- Chceme postup, ktorý konečným počtom krokov opíše ako problém vyriešiť ...
a týmto redukuje nekonečno na konečno.
- Algoritmus je jednoznačný konečný opis (postupu riešenia) problému.
- Často však mám problém (zadanie úlohy) a nemám (zatiaľ neviem) algoritmus ako ju riešiť...

David Hilbert a Kurt Gödel

- David Hilbert mal sen, že pre každý matematicky formulovateľný problém existuje algoritmus (konečný opis tohto problému) a ak ho nemáme je to naša hlúposť ... (a musíme len hľadať)
- Hilbert veril, že práca matematika (dokazovať) sa dá automatizovať
- Gödel, 1930 dokázal, že existujú matematické tvrdenia, ktoré sa nedajú dokázať ani vyvrátiť.
- Jazyk matematiky je silnejší na vyjadrovanie ako na argumentovanie (dokážem v ňom viac zapísať ako dokázať)

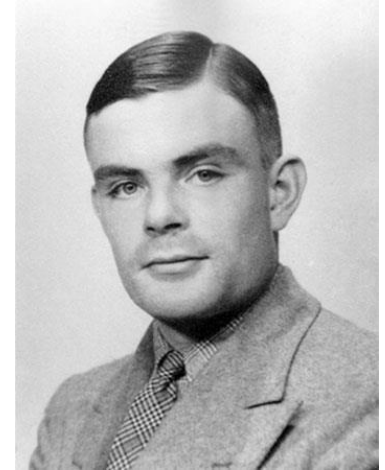


Vzniká teoretická informatika (computer science)

- Dokážem popísať algoritmy jednoznačne, a na vykonanie nepotrebujem človeka (jeho improvizáciu a vedomosti)
- Vznikla technológia, ktorá umožňuje mechanicky vykonávať postupnosti krokov: matematický výpočet môžem delegovať na stroje ...
- Predtým než boli prvé počítače ... chceme vedieť rozhodnúť, ktoré problémy sa dajú riešiť, a ktoré sa nedajú riešiť
- Aby sme to mohli matematicky dokázať, tak musíme vedieť matematicky zapísať algoritmus ...

Alan Turing

- Turing vymyslel **Stroj**, a všetko čo sa na ňom dá vykonať je algoritmus, ostatné nie...
- Prečo mu ľudia verili, že TOTO je ten stroj, ktorý definuje všetky algoritmy?
- Turing si uvedomil, že všetko (všetky informácie) sú postupnosti symbolov; matematik upravuje postupnosti symbolov ...
- Čo to znamená „vykonať operáciu“?
 - Vyberiem si postupnosť symbolov a vykonám nad ňou operáciu ...
 - dostanem inú postupnosť symbolov
- Dĺžka postupnosti nemôže byť obmedzená (dĺžka pásky je neobmedzená).



Alan Turing

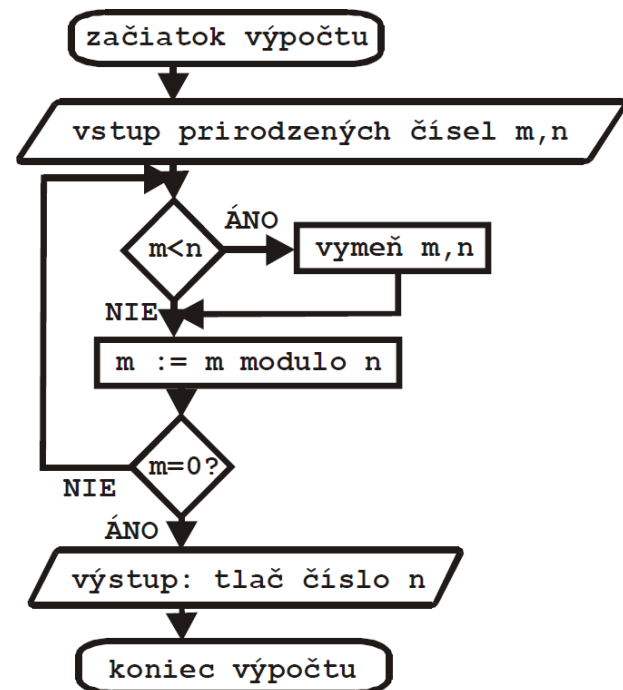
- Turing si uvedomil, že náš mozog je konečne veľký a teda z každého „nekonečna“ vidíme len konečne veľkú časť...
- Čiže zmena, ktorá môže vo výpočtovom stroji nastať, je najviac konštantne veľká (ako aj kapacita mozgu)
- Ktoré operácie teda umožním vykonávať, aby sa dalo vykonať „všetko“?
- **operácie, ktoré môžu zmeniť najviac jeden symbol**
- To sú všetky operácie, ktoré potrebujeme (zložitejšie vieme rozdeliť na jednotlivé kroky úprav po jednom symbole)
- Týmto sme definovali **Turingov stroj**
 - Je to to isté, ako keď dáme matematikovi ceruzku a gumu, tak potom mu stačí písať a gumovať na 1D páske

Alan Turing

- Týmto sme definovali **Turingov stroj**
 - Je to isté, ako keď dáme matematikovi ceruzku a gumu, tak potom mu stačí písať a gumovať na 1D páske
- Aj iné výpočtové formalizmy sa ukázali, že sú ekvivalentné s **Turingovým strojom**
- **Kvantový počítač (fyzikálna realita) je tiež ekvivalentný s Turingovým strojom**

Algoritmus

- Postup na vyriešenie úloh určitého typu
vstup → výstup
- Jednoznačnosť krokov
- Všeobecnosť použitia
- Vlastnosti
 - Konečnosť
 - Efektívnosť



Inými slovami

- **Algoritmus** v informatike je **jednoznačná, presná a konečná** postupnosť operácií, ktoré sú aplikovateľné na množinu objektov alebo symbolov (čísiel, šachových figúrok, ingrediencií na bábovku).
- Počiatočný stav týchto objektov je **vstupom**, ich koncový stav je **výstupom**.
- Počet operácií, vstupy a výstupy sú **konečné** (aj keď bežne počítame napr. s iracionálnym číslom π , vždy jeho číselnú reprezentáciu obmedzíme pri numerických výpočtoch na konečnú presnosť, napr. $\pi=3.14$).
- Jeden problém môže byť riešený viacerými algoritmami

Vlastnosti algoritmov

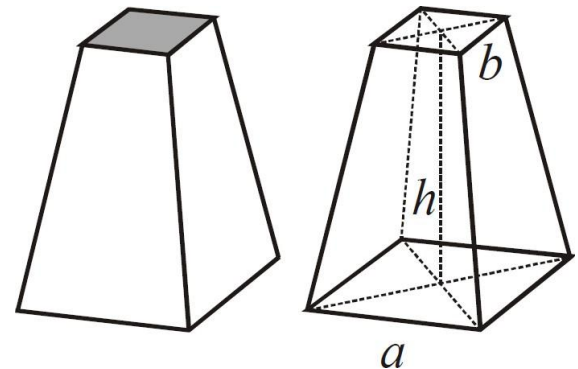
- **Jednoznačnosť** znamená, že každý krok algoritmu musí byť presne definovaný. Nesmie dovoliť viac výkladov, jednoznačne je určený krok za ním nasledujúci.
- **Univerzálnosť** algoritmu znamená, že je použiteľný pre riešenie veľkej skupiny úloh toho istého typu, líšiacich sa vstupnými údajmi (tak napr. algoritmus pre hľadanie koreňov kvadratickej rovnice je použiteľný pre KAŽDÚ kvadratickú rovnicu).

Vlastnosti algoritmov (2)

- **Rezultatívnosť** znamená, že algoritmus vždy musí po konečnom počte krokov dojsť k nejakému riešeniu.
- **Správnosť (korektnosť)** algoritmu znamená, že odpoveď pre každý vstup je správna a korektná - teda vyhovuje špecifikácii problému, ktorý algoritmus o sebe tvrdí, že rieši.
- **Efektívnosť** algoritmus zahŕňa zdroje potrebné na vykonanie výpočtu - výpočtový čas, kapacita pamäte, počet správ pri komunikácii, a pod.

Prvé algoritmy v Starom Egypte

- Návod na výpočet objemu zrezaného ihlanu (ihlan so štvorcovou podstavou, ktorého vrchol je zrezaný rovinou rovnobežnou s podstavou) podľa tzv. Moskovského papyrusu (Egypt, 1850 p.n.l.):
 - Jedaná orezaná pyramída, ktorej výška je 6, strana podstavy je 4 a strana hornej základne je 2. Vypočítaj objem tejto pyramídy:
 1. Umocni číslo 4 na druhú, dostaneš 16.
 2. Číslo 4 zdvojnásob, dostaneš 8.
 3. Umocni na druhou číslo 2, dostaneš 4.
 4. Tieto čísla 16, 8 a 4 sčítaj, dostaneš 28.
 5. Urči tretinu z čísla 6, dostaneš 2.
 6. Zdvojnásob číslo 28, dostaneš 56.
 7. Celkový výsledok je 56, počítal si dobre.
- Použitie symbolov až u starých Grékov, $V = \frac{1}{3} h (a^2 + ab + b^2)$
- **Formalizácia konceptu algoritmu v Euklidových Základoch** (3. st. p. n. l.)



Najväčší spoločný deliteľ prirodzených čísel m a n

- Euklidov algoritmus:
 1. Keď m je menšie ako n , vymeň ich hodnoty
 2. Do m daj hodnotu zvyšku po delení m/n (v modernej terminológii sa to zapisuje ako $m := m \bmod n$)
 3. Keď sa m nerovná nule, chod' na krok 1 s novými hodnotami m a n
 4. Vráť n ako výsledok

Eratosthenovo sito (3. st. p.n.l.)

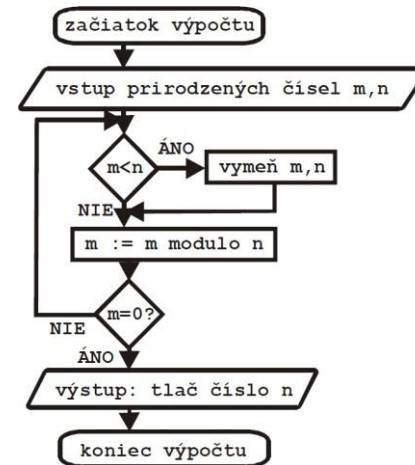
- Nájdienie všetkých prvočísel menších ako n (prvočíslo je celé číslo väčšie ako 1, ktoré je bezo zvyšku deliteľné iba sebou samým a číslom 1, v súčasnej dobe sa prvočísla často využívajú napr. pri šifrovaní správ). Eratosthenov predpis znie:
 1. Zapiš do zoznamu všetky čísla od 2 do n
 2. Nech $k=2$
 3. Pre každé číslo m medzi $k+1$ a n skontroluj, či je presným násobkom k , keď áno, vyškrtni m zo zoznamu.
 4. Do k daj najmenšie ešte nevyškrtnuté číslo zo zoznamu
 5. Keď k je menšie ako n , opakuj od kroku 3
 6. Akékoľvek číslo zo zoznamu, ktoré nebolo vyškrtnuté, je prvočíslo

	1	2	3	4	5	6	7	8	9	10
deliteľné	def.			2		2		2	3	2
	11	12	13	14	15	16	17	18	19	20
deliteľné		2		2	3	2		2		2
	21	22	23	24	25	26	27	28	29	30
deliteľné		2		2		2		2		2
	31	32	33	34	35	36	37	38	39	40
deliteľné		2	3	2	5	2		2	3	2
	41	42	43	44	45	46	47	48	49	50
deliteľné		2		2	3	2		2	7	2

Zápis algoritmus

- Zrozumiteľnosť a stručnosť zápisu
- Slovný opis
- Vývojový diagram
- Pseudokód
- Kód v programovacom jazyku
- Logický obvod

Vstup dvoch celých čísel: m, n
rob
 keď $m < n$ vymeň m, n
 $m := m \bmod n$
dokiaľ $n \neq 0$
vytlač n

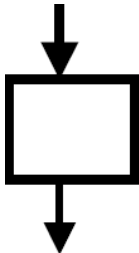


```
int nsd(int m, int n)
{
    if (n == 0)
        return m;
    return nsd(n, m%n);
}
```

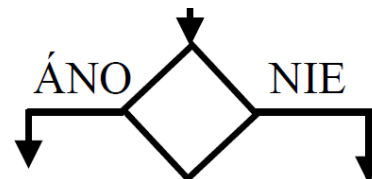
Jazyk vývojových diagramov

- Popisuje tok riadenia od jedného k ďalšiemu kroku algoritmu (bežne zhora dole) pomocou orientovaných hrán (šípok) spájajúcich činnosti opísané v blokoch.

- Začiatok, resp. koniec 

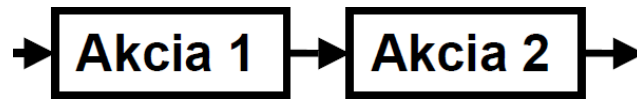
- Operačný blok, obsahuje akcie 

- Rozhodovací blok, splnenie/nesplnenie podmienky určí nasledujúci krok riadenia

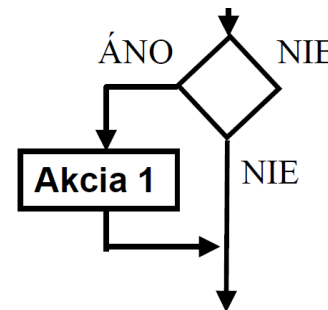
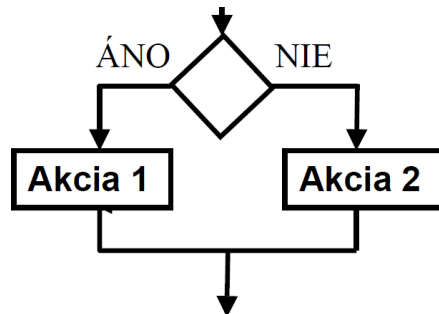


Jazyk vývojových diagramov (2)

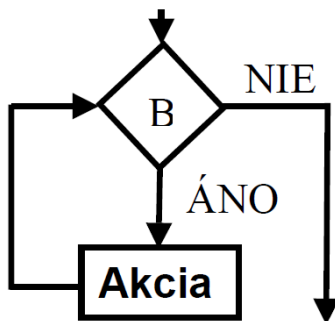
- Sekvencia



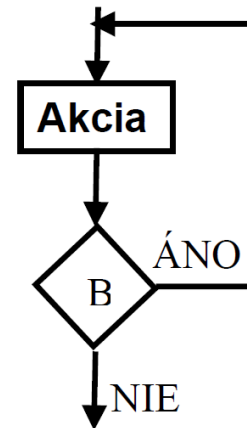
- Vetvenie



- Cyklus s testom na začiatku



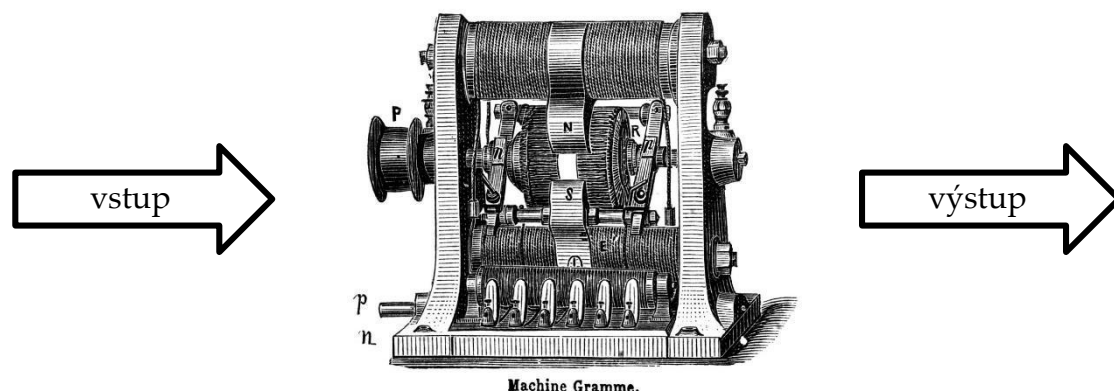
- Cyklus s testom na konci



Dátová štruktúra (opakovanie)

- Spôsob organizácie a uchovania dát, ktorý umožňuje efektívne využitie /spracovanie dát
- Operácie - čo umožňuje s dátami robiť
- Konzistentnosť
- Vlastnosti
 - Súbežnosť (concurrency)
 - Perzistentnosť (persistency)
 - Dynamickosť (online/offline)
 - Efektívnosť

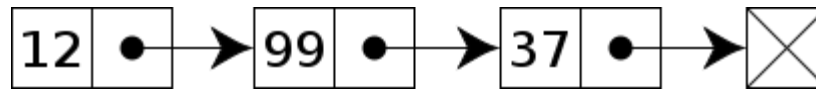
Príklad dátovej štruktúry



- Načo je dobrá?
- Operácie, ktoré s ňou môžeme realizovať:
 - Vlož prvok do množiny
 - Odstráň prvok z množiny
 - Zisti, či prvok je v množine?
- Technická realizácia (implementácia) je druhoradá

Možná implementácia

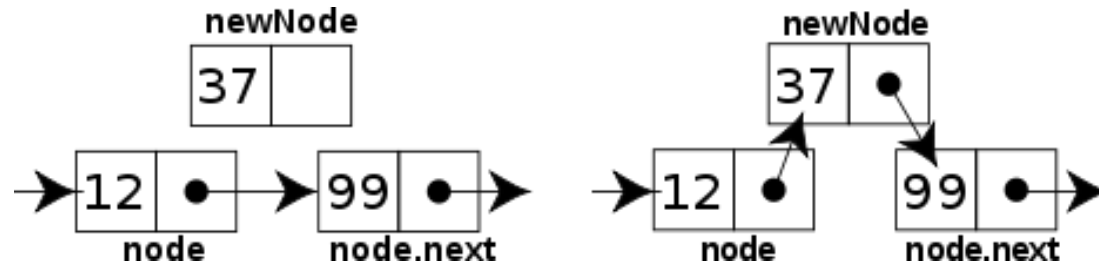
- Spájaný zoznam (linked list)
- Jednosmerne zret'azený



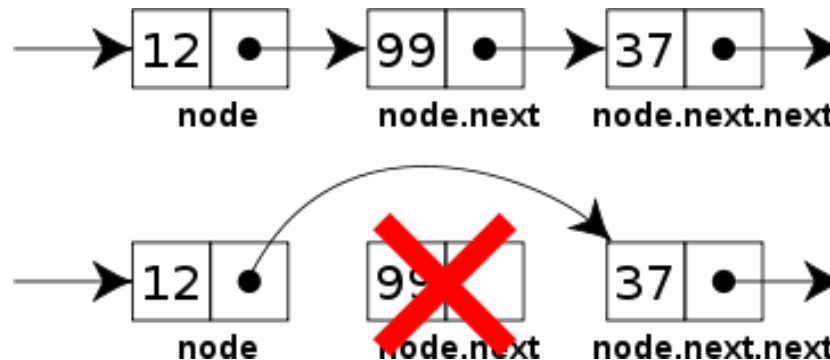
- Prvky množiny vkladané do reťaze za sebou
- Vloženie prvku do množiny realizujeme ako vloženie prvku do reťaze
- Odstránenie prvku z množiny realizujeme ako odstránenie prvku z reťaze
- Zistenie, či prvok je v množine realizujeme ako vyhľadanie prvku v reťazi

Spájaný zoznam - operácie

- Vloženie (operácia insert)

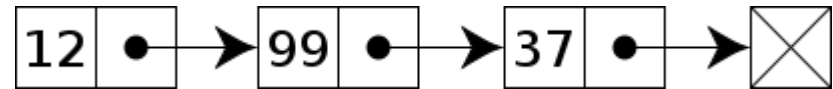


- Odstránenie (operácia remove)

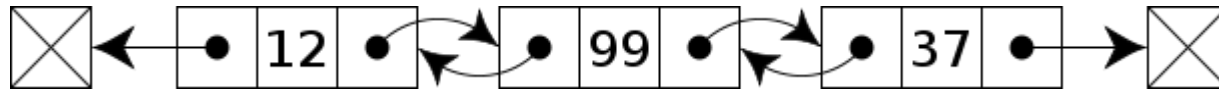


Iné varianty spájaného zoznamu

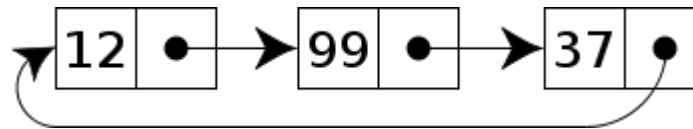
- Jednosmerne zret'azený



- Obojsmerne zret'azený



- Cyklický



- Pomocné prvky (tzv. sentinel /dummy nodes)
- Ako vyzerá prázdny zoznam?
- Ako zistíme, či je zoznam cyklický?

Ret'azec / pole (vektor)

40	30		10	20	-5	30		
0	1	2	3	4	5	6	7	8

■ Operácie:

- Nastaviť i-ty prvok - **Set(i, value)**
- Zistiť hodnotu i-teho prvku - **value** \leftarrow **Get(i)**
resp. **getCharAt(i)**
- Nájsť pozíciu prvku - **index** \leftarrow **find(value)**
- Určenie dĺžky - **uint** \leftarrow **getLength()**
- Je prázdny? - **bool** \leftarrow **isEmpty()**

Jeden problém, viac riešení

- Problém je zaujímavý vtedy, keď je zadaný všeobecne, aby pokrýval rozmanitosť problémov reálneho sveta
- Všeobecne zadaný problém má zvyčajne viacero možných riešení
- Chceme vedieť rozlíšiť rôzne tieto riešenia, a zvoliť to, ktoré vyhovuje požiadavkám našej situácie

Pre dané N určit súčet čísel 1, 2, ..., N

- Prvé možné riešenie:
čísla postupne pripočítavame k výsledku

```
int i, sucet = 0;
for (i = 1; i <= N; i++)
    sucet += i;
```

- Vykonáme rádovo N operácií
 - Pre väčšie vstupy (zvyšujúce sa N) sa úmerne zvyšuje aj výpočtová zložitosť

Pre dané N určit súčet čísel 1, 2, ..., N

- Druhé možné riešenie:
výpočet použitím vzorca v uzavretom tvare

```
int sucet = N*(N+1)/2;
```

- Vykonáme vždy konštantný počet operácií
 - Pre väčšie vstupy (zvyšujúce sa N) sa **NEZVYŠUJE** výpočtová zložitosť

Prečo má zmysel analyzovať algoritmy?

- Klasifikovať problémy a algoritmy podľa náročnosti
- Predvídať výkon, porovnávať algoritmy a dátové štruktúry, vylad'ovať parametre
- Lepšie porozumieť a vylepšovať implementácie algoritmov a dátových štruktúr
- **Intelektuálna výzva**

Alan Turing (1947) - Donald E. Knuth (1960)



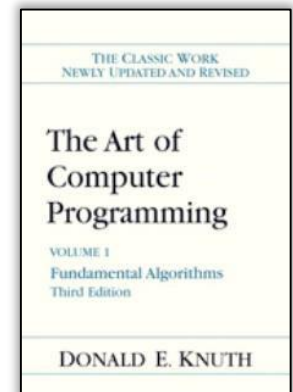
"It is convenient to have a measure of the amount of work involved in a computing process, even though it be a very crude one. We may count up the number of times that various elementary operations are applied in the whole process ..."

— Alan Turing (1947)

■ Donald E. Knuth:

To analyze an algorithm:

- Develop a good implementation.
- Identify unknown quantities representing the basic operations.
- Determine the cost of each basic operation.
- Develop a realistic model for the input.
- Analyze the frequency of execution of the unknown quantities.
- Calculate the total running time: $\sum_q \text{frequency}(q) \times \text{cost}(q)$



Súčasnost'

- Aho, Hopcroft, Ullman (1970),
- Cormen, Lieserson, Rivest, Stein (1990-)
- Analýza najhoršieho prípadu
- Použitie O-notácie pre asymptotický horný **odhad**
- Klasifikujeme algoritmy podľa týchto zložítostí
- Nevýhoda prístupu odhadovania zložítostí algoritmov:
Nemôžeme použiť na predvídanie výkonu alebo priame porovnanie algoritmov!
 - Quicksort - počet porovnaní v najhoršom prípade $O(N^2)$
 - Mergesort - počet porovnaní v najhoršom prípade $O(N \log N)$
 - V praxi je však Quicksort zvyčajne dva krát rýchlejší a používa polovičné množstvo pamäti...

Ako merať zložitosť algoritmov?

- Analýza zložitosti algoritmu je **výpočet-odhad** požiadaviek na výpočtové prostriedky, ktoré bude vykonanie algoritmu vyžadovať **v závislosti na veľkosti vstupu**
- Veľkosť vstupu
 - Počet bitov vstupného čísla
 - Dĺžka postupnosti čísel na vstupe
 - Rozmery vstupnej matice
 - Počet znakov textu na vstupe
 - ...

Výpočtové prostriedky

- Výpočtová (časová) zložitosť
 - Cykly procesora
 - Doba výpočtu
 - Počet vykonaných inštrukcií
 - Počet transakcií nad databázou
- Pamäťová (priestorová) zložitosť
 - Pamäťové bunky
- Komunikačná zložitosť
 - Sieťová kapacita
 - Počet spojení
- Algoritmus môže byť tzv. **citlivý na vstup** (na hodnoty vstupu, nielen množstvo vstupu)

Model výpočtového systému

- Jednoprocesorový Random Access Machine (RAM)
- Dáta sú uložené v adresovateľnej pamäti
- Vykonáva aritmeticko-logické, riadiace a pamäťové inštrukcie
- Časová zložitosť inštrukcií je jednotková (konštantná)
 - Rozšírenie: môžeme uvažovať aj váhu (cenu) inštrukcií
- Cykly a volanie funkcií nie sú jednoduché operácie a ich zložitosť závisí na veľkosti dát a obsahu funkcie
- Inštrukcie sú vykonávané postupne-sekvenčne
 - Neuvažujeme súbežné vykonávanie vlákien
 - rozšírenie: Parallel Random Access Machine (PRAM)

Analýza prípadov

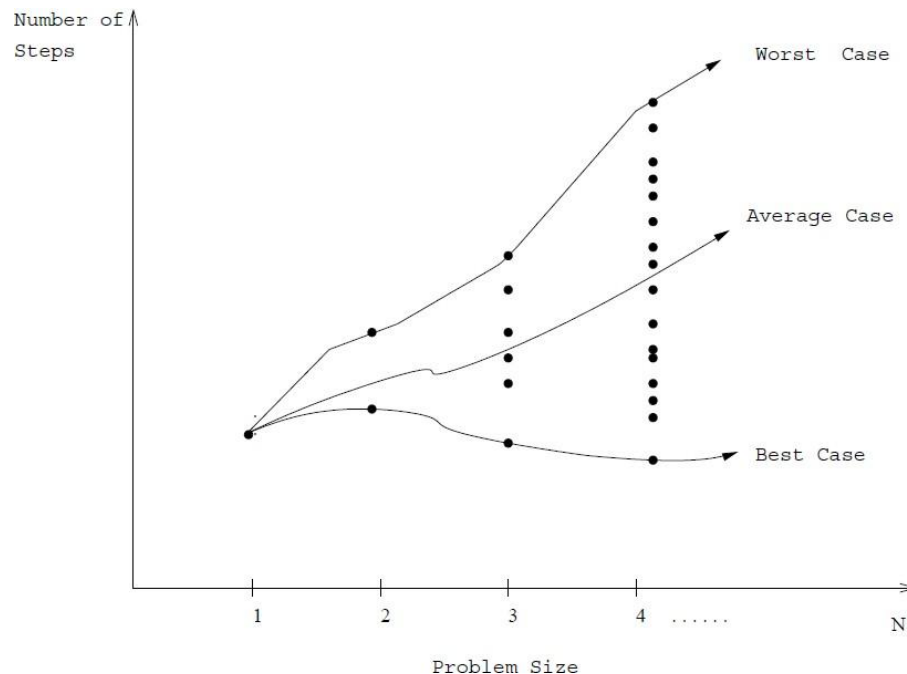
- Aký je rozdiel medzi týmito prípadmi?
 - Najhorší prípad algoritmu
 - Najlepší prípad algoritmu
 - Priemerný prípad algoritmu

Vstup je veľkosti N

Analýza prípadov

- **Najhorší prípad (worst-case)**

Zložitosť algoritmu v najhoršom prípade je funkcia definovaná maximálnym počtom krokov, ktoré algoritmus vykoná pri (ľubovoľnom) vstupe veľkosti N .
→ obzvlášť dôležité pre real-time systémy



Analýza prípadov (2)

- **Najlepší prípad (best-case)**

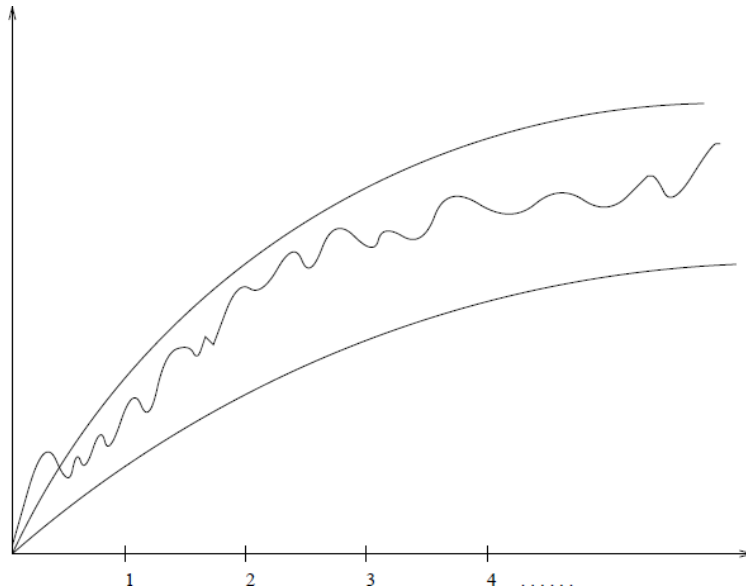
Zložitosť algoritmu v najlepšom prípade je funkcia definovaná minimálnym počtom krokov, ktoré algoritmus vykoná pri (nejakom) vstupe veľkosti N .

- **Priemerný prípad (average-case)**

Zložitosť algoritmu v priemernom prípade je funkcia definovaná priemerným počtom krokov, ktoré algoritmus vykoná pri vstupoch veľkosti N .

Asymptotická zložitosť

- Presné určenie počtu vykonaných operácií:
 - veľmi náročné v prípade zložitých algoritmov
 - skoro zbytočné pre jednoduché algoritmy
- Jednoduchšie je analyzovať horné a dolné ohraničenia počtu vykonaných krokov



Asymptotická zložitost' (2)

- Prakticky nás zaujíma ako sa bude algoritmus správať pre veľké vstupy idúce do nekonečna
- Vyjadrujeme rád rastu funkcie, zanedbávame príspevok nižších rádov
- Asymptotická zložitost' je vyjadrenie pre takú (veľkú) veľkost' problému, aby sa prejavil rád rastu funkcie zložitosti v závislosti na veľkosti vstupu
- Asymptoticky lepší algoritmus bude lepší pre všetky vstupy okrem konečného počtu malých vstupov

Asymptotická dominancia

n	$f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10		0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20		0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30		0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40		0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50		0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100		0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000		0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000		0.013 μs	10 μs	130 μs	100 ms		
100,000		0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000		0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000		0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000		0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000		0.030 μs	1 sec	29.90 sec	31.7 years		

Zhrnutie

- Algoritmus
- Dátová štruktúra
- Analýza zložitosti algoritmov
- Jeden problém, viac riešení
 - Súčet čísel $1+2+\dots+N$: výpočet sumy vs vzorec
 - Dátová štruktúra pre množinu: spájaný zoznam vs vektor
- Použitie O-notácie pre asymptotický horný odhad
- Bodovanie v predmete:
Zadania (40b) + Skúška (60b)