Ing. Peter Jankovič, PhD.

Algoritmy a údajové štruktúry

2

Význam používania vhodných dátových štruktúr, ich prehľad a systemizácia. Efektívne využívanie údajových štruktúr pri návrhu aplikácií.

Požiadavky na absolvovanie

Aby zložil študent skúšku, musí splniť tieto podmienky:

- úspešne odovzdať obe zadané semestrálne práce
- z hodnotenia počas semestra získať aspoň 30 bodov z 50 možných,
- z hodnotenia na skúške získať aspoň 13 bodov z písomnej časti (celkovo 25 bodov) a aspoň 13 bodov z ústnej časti (celkovo 25 bodov),
- získať celkovo aspoň 61 bodov zo 100 možných.

Podmienkou pre úspešné absolvovanie cvičení k predmetu je systematická práca na cvičeniach posudzovaná podľa počtu nazbieraných bodov za priebežné testy a semestrálne práce.

Hodnotenie

- A → 93 – 100 bodov
- B → 85 – 92 bodov
- C → 77 – 84 bodov
- D → 69 – 76 bodov
- E → 61 – 68 bodov
- Fx → menej než 61 bodov

Plán prednášok

1. **Význam používania vhodných dátových štruktúr, ich prehľad a systemizácia. Efektívne využívanie údajových štruktúr pri návrhu aplikácií.**
2. **Typy vyhľadávaní dát. Vyhľadávacie stromy a ich implementácia: AVL strom, 2-3 strom, 2-4 strom, splay strom, treap.**
3. **Vyhľadávacie stromy a ich implementácia: RB strom, AA strom, Tango strom, znakové stromy. Štruktúra Skip list.**

Plán prednášok

4. Štruktúry uložené v operačnej pamäti podporujúce jednorozmerné intervalové vyhľadávanie - intervalové stromy. Štruktúry uložené v operačnej pamäti podporujúce viacrozmerné intervalové vyhľadávanie.
5. Pokročilé implementácie prioritného frontu: Fibonacciho halda, párovacia halda...
6. Organizácia externých pamätí a súborov na externých médiách. Sekvenčný súbor. Súbory s priamym prístupom. Štruktúra heap file.
7. B-strom. Efektívne triedenie dát v súboroch. Súvislý utriedený súbor. Indexované utriedené štruktúry: indexsekvenčný súbor, B+ strom. Indexované neutriedené štruktúry: súbor s úplným indexom.

Plán prednášok

8. **Statický hešovací súbor. Hešovacie súbory: rozšíriteľné hešovanie, dynamické hešovanie, lineárne hešovanie.**
9. **Štruktúry uložené v súboroch podporujúce viacrozmerné bodové vyhľadávanie. Štruktúry uložené v súboroch podporujúce viacrozmerné intervalové vyhľadávanie.**

Plán prednášok

10. Vyhľadávanie reťazcov v texte. Komprimačné algoritmy.
11. Poradové štatistiky.
12. Geometrické algoritmy.

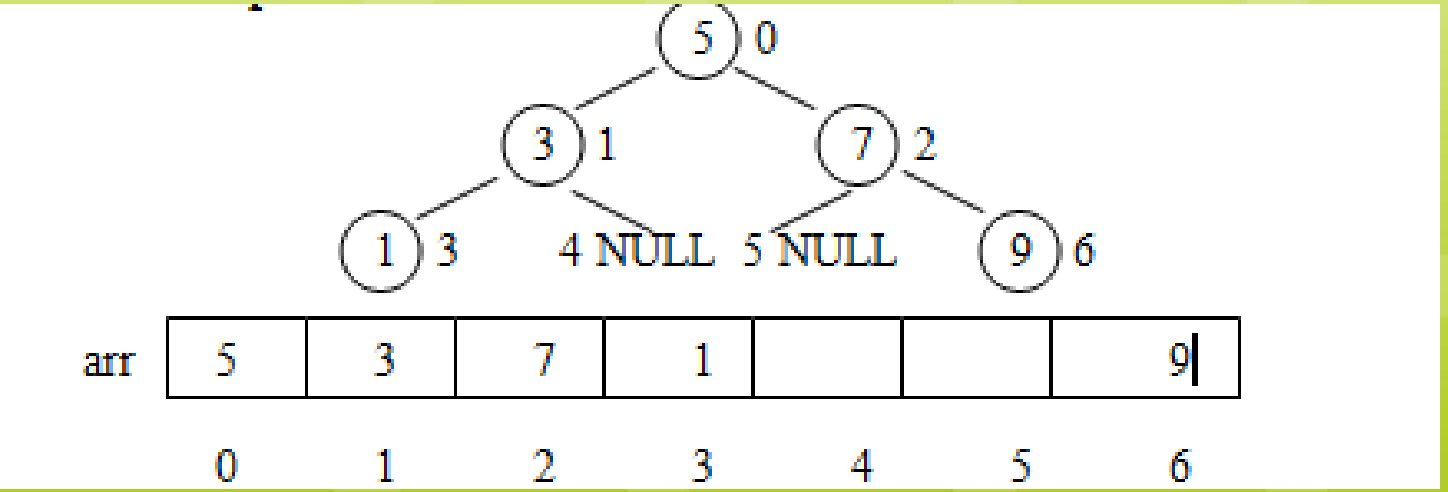
Význam používania vhodných ÚŠ

- rýchlosť aplikácie
  - šetrenie nákladov za hardvér
  - šetrenie času užívateľa
- pamäťová náročnosť aplikácie
  - šetrenie nákladov za hardvér
- rýchlejší vývoj aplikácie
  - šetrenie času programátora

Hardvér nikdy nesplní všetky nároky.

Ako vybrať vhodnú ÚŠ?

- Kde budú dáta uložené?



Zdroj obrázku: http://www.manojagarwal.co.in/images/bst1.png

Ako vybrať vhodnú ÚŠ?

- Kde budú dáta uložené?
- Pri väčšine prípadov je potrebné rozhodnúť, nakoľko je dôležitá rýchlosť a nakoľko pamäťová náročnosť.
- Pre vyhodnotenie zložitosti je potrebné definovať objektívne kritériá umožňujúce vzájomné porovnanie:
  - Čas potrebný k vykonaniu algoritmu – časová zložitosť algoritmu.
  - Veľkosť pamäťového priestoru potrebného k realizácii algoritmu – pamäťová zložitosť algoritmu.

Časová zložitosť algoritmu

Funkcia	Názov	n=10	n=100	n=1000	n=10000
O(1)	Konštantná zložitosť	1	1	1	1
O(n)	Lineárna zložitosť	10	100	1000	10000
$O(n^2)$	Kvadratická zložitosť	100	10000	10^6	10^8
$O(n^k)$	Polynomiálna zložitosť	1000	10^6	10^9	10^{12}
O(log(n))	Logaritmická zložitosť	1	2	3	4
O(n*log(n))	„Linearitmická“ zložitosť	10	200	3000	40000
$O(a^n)$	Exponenciálna zložitosť	1024	$\approx 10^{31}$	$\approx 10^{310}$	$\approx 10^{3100}$

V príklade k = 3 (kubická zložitosť), a = 2.

Zložitosť – najhoršia, priemerná.

Ako vybrať vhodnú ÚŠ?

- **Zvyčajne sa primárne snažíme minimalizovať časovú zložitosť, následne pamäťovú náročnosť.**

Ako implementovať?

- Správne pochopiť fungovanie údajovej štruktúry a jej výhody/nevýhody.
- Výber spôsobu implementácie (expilictná, implicitná, ...).
- Implementácia s dôrazom na rýchlosť spracovania operácií.
- Testovanie správneho fungovania a hľadanie výkonových problémov.

Uloženie údajovej štruktúry

- v operačnej pamäti
- na pevnom disku (HDD)
- na SSD disku (Solid-state drive)

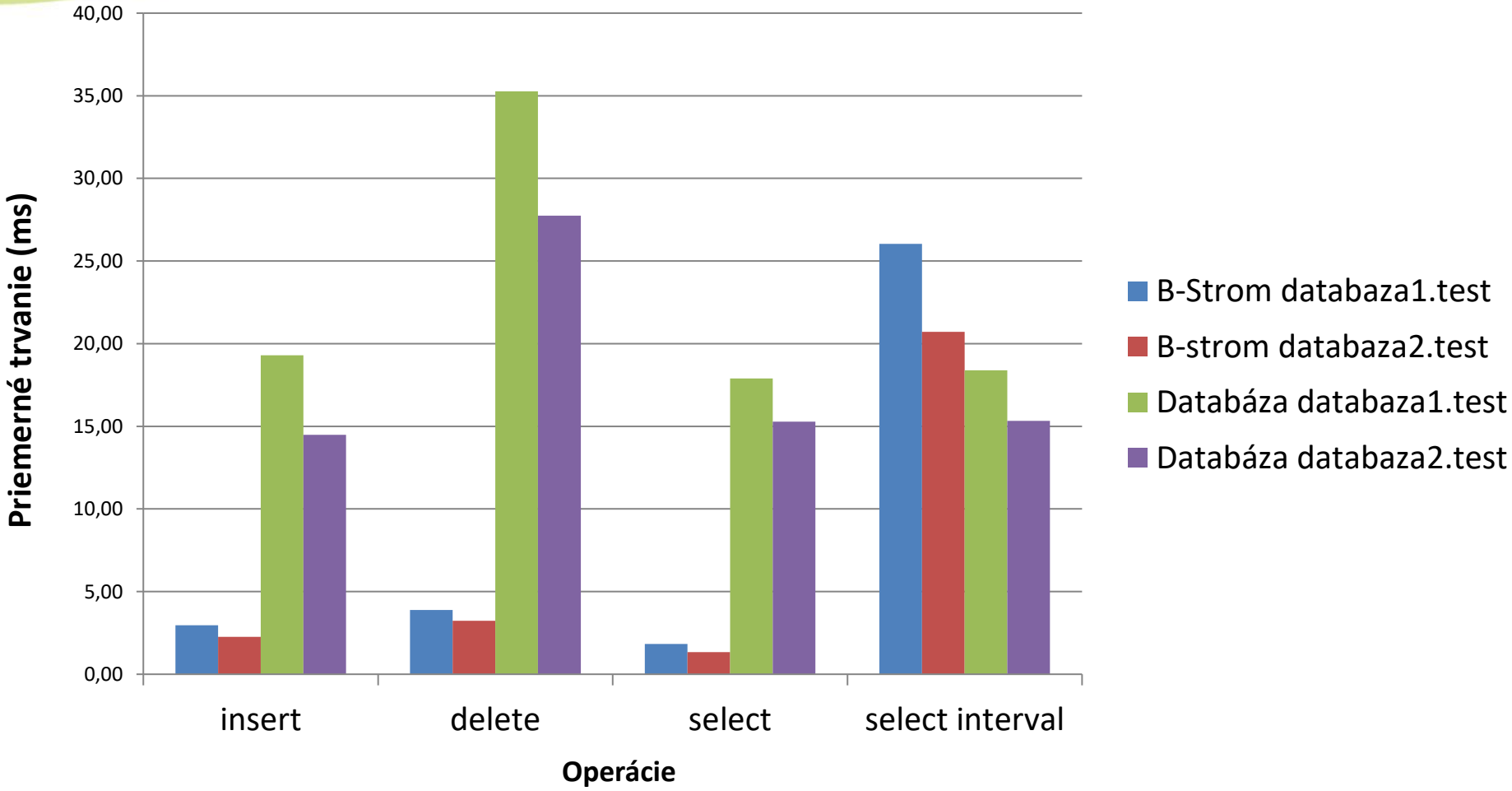
Implementácia údajovej štruktúry

- Explicitná – pomocou smerníkov/referencii
- Implicitná – pomocou poľa
- Od implementácie závisia aj niektoré vlastnosti a rýchlosť údajovej štruktúry.
- Pridanie prvku do zoznamu implementovaného pomocou dynamického poľa môže trvať podstatne dlhšie ako do lineárneho zreťazeného zoznamu

Memory manager

- Fyzická pamäť je mapovaná do virtuálneho adresného priestoru. Pri vytváraní nového objektu nie je potrebné zabezpečiť manažment pamäte o to sa postará správca pamäte.
- Neznalosť spôsobu uloženia údajovej štruktúry v pamäti vedie k jej nevhodnému používaniu.

1.Praktický príklad použitia ÚŠ



Zdroj: Židek V.: Experimentálne porovnanie výkonnosti databázového systému a údajovej štruktúry, 2012

1. Praktický príklad použitia ÚŠ

- Na vymazanie prvku v B-strome je potrebné len 11% času potrebného na vymazanie prvku pri použití databázového systému firmy Oracle s embedded driverom.
- Vo všeobecnosti sa databázové systémy optimalizujú pre operáciu select.
- Údajová štruktúra môže byť prispôsobená pre konkrétny problém.

2. Praktický príklad použitia ÚŠ

- V aplikácií potrebujeme evidovať autá (resp. čísla kariet majiteľov áut), ktoré stoja na parkovisku s cieľom zabrániť opakovanému použitiu karty a štatistickému sledovaniu vyťaženosti parkoviska
- Môžem v jazyku Java použiť napr.: ArrayList, LinkedList, Hashtable

2. Praktický príklad použitia ÚŠ

	Vloženie EČV	Vymazanie EČV	Rýchlosť	Možné spomalenie	Čas potrebný na alokáciu pamäte
ArrayList	O(1)	O(n)	vysoká	zväčšovanie dynamického poľa - int initialCapacity	dlhý
LinkedList	O(1)	O(n)	nízka	žiadne	krátky
Hashtable	O(1)	O(1)	vysoká	zväčšovanie dynamického poľa - int initialCapacity	dlhý

V tabuľke sú použité priemerné zložitosti.

3. Praktický príklad použitia ÚŠ

Pre potreby optimalizácie siete záchranných staníc v SR bola vytvorená aplikácia, ktorá dokáže analyzovať anonymizované záznamy zdravotných poisťovní o výjazdoch sanitných vozidiel záchranného systému. Dataset obsahuje celkovo 1 464 305 záznamov. Aplikácia, ktorá má také množstvo záznamov načítať a analyzovať musí byť navrhnutá s ohľadom na čo najlepšie využitie operačnej pamäte a rýchlosti spracovania údajov. Výsledkom analýzy sú údaje o priestorovom rozdelení pacientov, vekovej a pohlavnej štruktúre, časových trvaniach jednotlivých činností (doba ošetrenia pacienta na mieste zásahu, doba potrebná na odovzdanie pacienta v zdravotníckom zariadení a pod.) a mnoho iných vyhodnotení.

3. Praktický príklad použitia ÚŠ

- Pre účely testovania rôznych štruktúr bola aplikácia upravená. Hlavná údajová štruktúra, ktorá je použitá na mnohých miestach v programe sa vždy zmenila tak, aby bolo možné porovnať rýchlosť.
- Pri profilovaní bola použitá metóda vzorkovania. Tabuľka uvádza počet vzoriek, ktoré boli zaznamenané pre jednotlivé metódy. Pre účely porovnania rýchlosti nám takéto vyhodnotenie postačuje.

3. Praktický príklad použitia ÚŠ

Štruktúra	Počet vzoriek
ArrayList – neprealokovaný	70 292
ArrayList – prealokovaný	69 944
FastUtil ArrayList – neprealokovaný	68 022
FastUtil ArrayList – prealokovaný	68 004
LinkedList	69 566
TreeSet	87 536
TreeMap	357
LinkedHashMap – neprealokovaná	285
LinkedHashMap – prealokovaná	276
HashMap – neprealokovaná	334
HashMap – prealokovaná	329
Commons collections HashedMap – neprealokovaná	829
Commons collections HashedMap – prealokovaná	247

3. Praktický príklad použitia ÚŠ

- Pri použití štruktúry ArrayList z knižnice java.util môžeme vidieť, že načítanie a spracovanie záznamov trvá v porovnaní s inými štruktúrami skutočne dlho (zaznamenalo sa najviac vzoriek). Taktiež vidíme, že ak nastavíme inicializačnú kapacitu v konštruktore (ArrayList – prealokovaný), dôjde k miernemu zlepšeniu rýchlosti celého spracovania údajov.
- Pri použití štruktúry ArrayList z knižnice FastUtil môžeme vidieť, že načítanie a spracovanie záznamov trvá kratší čas (zaznamenalo sa menej vzoriek). Taktiež vidíme, že ak nastavíme inicializačnú kapacitu v konštruktore tak, dôjde k miernemu zlepšeniu rýchlosti celého spracovania údajov. Tu je dôležité všimnúť si aj porovnanie so štruktúrou ArrayList z knižnice java.util. Vidíme, že aj rovnaká štruktúra sa môže líšiť v rýchlosti v závislosti na kvalite jej implementácie.

3. Praktický príklad použitia ÚŠ

- Mnohé štruktúry sú pre zrýchlenie svojho fungovania podporené ďalšími pomocnými štruktúrami. Vhodným príkladom je použitie štruktúry TreeSet z knižnice java.util, čo je TreeMap obohatený o usporiadaný zoznam. Keďže v našom prípade usporiadaný zoznam, ktorý táto štruktúra obsahuje nikde nevyužijeme dôjde k zbytočnému výraznému spomaleniu voči použitiu štruktúry TreeMap z knižnice java.util, ktoré je spôsobené tvorbou usporiadaného zoznamu.
- Použitie štruktúry LinkedHashMap z knižnice java.util celé spracovanie značne urýchľuje, navyše ak sa správne nastaví aj inicializačná kapacita, môže byť zrýchlenie ešte väčšie. Keďže ide o kombináciu HashMapy a linerárneho zoznamu, je urýchlené prechádzanie všetkých prvkov v štruktúre a v našom prípade je táto kombinácia rýchlejšia ako použitie štruktúry HashMap z knižnice java.util .

3. Praktický príklad použitia ÚŠ

- Pri testovaní štruktúry HashedMap z knižnice Commons collections môžeme vidieť, že bez správnej inicializácie je štruktúra výrazne pomalšia voči iným štruktúram založeným na hešovacej tabuľke. Pri správnom nastavení inicializačnej kapacity sa spracovanie údajov výrazne urýchlilo a je to najlepší výsledok nášho testu.

3. Praktický príklad použitia ÚŠ

- Môžeme teda konštatovať, že znalosť údajových štruktúr, ich správne použitie a ich vhodná voľba je nutnou podmienkou pre vytvorenie kvalitnej aplikácie, ktorá pracuje s väčším množstvom údajov.

4. Praktický príklad použitia ÚŠ

V simulačnom modeli záchrannej zdravotnej služby (ZZS) SR je potrebné zabezpečiť nájdenie najrýchlejšej cesty. Najskôr sa náhodne vygeneruje poloha pacienta so zohľadnením hustoty obyvateľstva. Následne je potrebné k tomuto pacientovi vyslať vozidlo ZZS, ktoré je voľné a má najkratší dojazdový čas. Po príchode posádky ZZS k pacientovi prebehne jeho ošetrenie. Potom sa posádka buď vracia späť na stanicu a už počas tejto jazdy môže byť pridelená inému pacientovi, alebo vezie pacienta do nemocnice. Po príchode do nemocnice prebehne odovzdanie pacienta a posádka sa vracia späť na stanicu. Rýchlosť práce simulačného modelu je závislá na tom ako rýchlo dokážeme nájsť najrýchlejšiu cestu k pacientovi a prípadne aj do nemocnice. Cestná sieť je modelovaná na základe údajov z OpenStreetMap a obsahuje 1 020 848 vrcholov. Pre hľadanie najrýchlejšej trasy sa využíva Dijkstrov algoritmus.

4. Praktický príklad použitia ÚŠ

- Pred každým spustením algoritmu je potrebné všetky vrcholy inicializovať. Väčšinou sa používa „značka“ (premenná), ktorá určuje, aká je aktuálna najkratšia vzdialenosť od štartovacieho vrcholu až k tomuto vrcholu. Čiže pred štartom algoritmu sa musia všetky vrcholy uložené v nejakej štruktúre prejsť a správne inicializovať (nastaviť im značku na veľkú hodnotu). Výsledky pre viac ako 14 000 vygenerovaných pacientov môžeme vidieť v tabuľke.

Štruktúra	Priemerné trvanie [ms]
HashMap – prealokovaná	629
LinkedList	443
ArrayList – prealokovaný	364
BitSet	139

4. Praktický príklad použitia ÚŠ

- Vidíme, že ak sú vrcholy uložené v štruktúre HashMap, je celkové priemerné trvanie nájdenia cesty najpomalšie. To je spôsobné tým, že táto štruktúra nie je primárne určená na prechádzanie všetkých prvkov. Pri použití štruktúry LinkedList je situácia lepšia. Štruktúra ArrayList sa podľa predpokladov ukázala ako najlepšia, teda prejdenie všetkých prvkov v tejto štruktúre je najrýchlejšie.
- Navrhli a otestovali sme ale ešte aj jeden úplne iný spôsob inicializácie práce algoritmu. Nie je podstatné, v akej štruktúre budú vrcholy uchovávané. Vrcholy sa pred štartom algoritmu vôbec neinicializujú, ich „značky“ sa nemenia. Máme zadeklarované bitové pole (BitSet z knižnice java.util) s takým počtom bitov, ako je počet vrcholov, pričom každý vrchol má trvalo pridelený index. Pred štartom algoritmu sa inicializuje iba tento BitSet zavolaním metódy BitSet.clear(). Nie je teda potrebné prechádzať 1 020 848 inštancií tried a pre každú nastavovať hodnotu atribútu.

4. Praktický príklad použitia ÚŠ

- Inicializácia BitSetu je mnohonásobne rýchlejšia. Počas práce algoritmu potom stačí vždy pred „použitím“ vrcholu skontrolovať, hodnotu bitu na príslušnom indexe vrcholu. To vykonáme pomocou metódy BitSet.get(int bitIndex). Ak je tam nula (hodnota false), tak vrchol ešte nebol inicializovaný, teda sa nainicializuje sa a v BitSete sa nastaví na tomto indexe hodnota 1 zavolaním metódy BitSet.flip(int bitIndex). Týmto spôsobom sa budú vždy počas práce algoritmu inicializovať iba vrcholy, ktoré sa aj skutočne použili. Tento prístup zohľadňuje skutočnosť, že vozidlo ZZS sa pre pacienta z Košíc nebude posielať z Bratislavy a vyhľadávanie sa realizuje vždy na „malom“ lokálnom území (na Slovensku máme 273 staníc ZZS).
- Na základe výsledkov experimentu môžeme vidieť, že aj triviálna vec, akou je inicializácia vrcholov pre Dijkstrov algoritmus, sa pri sofistikovanom využití údajových štruktúr môže výraznou mierou podieľať na zrýchlení práce celého algoritmu.

Výhody použitia ÚŠ

- Údajová štruktúra môže byť optimalizovaná pre konkrétny problém, čím môže byť rádovo rýchlejšia a pamäťovo efektívnejšia. Mnohé problémy by neboli bez vhodných ÚŠ riešiteľné v reálnom čase.
- Odpadajú problémy s neskoršou kompatibilitou externých produktov.
- Nízke obstarávacie náklady, žiadne náklady za „updaty“.

Údajové štruktúry, ktoré poznáte

- pole
- zoznam
- hešovacia tabuľka
- strom, binárny vyhľadávací strom
- ...

Zoznam a jeho varianty

- Najčastejšie je implementovaný dynamickým poľom, alebo pomocou zreťazených prvkov.
- jednosmerný, dvojsmerný
- LIFO (Last in first out) – tiež nazývaný zásobník (stack) – operácia ***vyber*** vyberie naposledy vložený prvok
- FIFO (First in first out) – tiež nazývaný Front – operácia ***vyber*** vyberie prvok, ktorý sa nachádza vo fronte najdlhšie
- prioritný front – prvky sú vyberané podľa priority

Dostupné implementácie zoznamu

Java	C#	Vlož	Vymaž	Prístup k prvku na indexe	Poznámka
ArrayList	ArrayList, List	O(1)	O(n)	O(1)	Nevhodný pre časté volanie Vlož (dá sa eliminovať nastavením kapacity) a Vymaž. Zmenšovanie celého poľa pri mazaní prvku, posun indexov (c#).
LinkedList	LinkedList	O(1)	O(n)	O(n)	Nie je potrebná práca s dynamickým poľom. Minimálne nároky na pamäť.

V tabuľke sú použité priemerné zložitosti.

Prístup podľa kľuča

Java	C#	Vlož	Vymaž	Nájdi kľuč	Prístup k prvku na indexe	Poznámka
	SortedList	O(n)	O(n)	O(log n)	O(1)	Implementované dynamickým poľom.
Hashtable	Hashtable, Dictionary	O(1)	O(1)	O(1)	nepodporuje	Dôležité je vhodne nastaviť kapacitu. V jave je nastaviteľný loadFactor.

V tabuľke sú použité priemerné zložitosti.

Vizualizácie

- V rámci projektovej výučby vznikla knižnica viacerých štruktúr spolu s ich vizualizáciou. Berte prosím na vedomie, že ide o študentskú prácu, ktorá je len pomôckou pre Vás. Prípadné chyby môžete nahlásiť cez fórum. Textová časť neslúži k príprave na skúšku a pri implementácii je lepšie používať texty z prednášok.

http://frdsa.fri.uniza.sk/~jankovic/WEB/index.html

Skúsenosti

- Predmet je náročnejší v porovnaní s obsahom bakalárskeho štúdia na FRI INF, ale nie nezvládnuteľný.
- Predmet bol narocny, ale netvrdil by som, ze zbytocne. Skor uzitocne.
- Nie je to predmetom ale skôr nedostatočnou predošlou prípravou na iných predmetoch. Množstvo študentov je slabo pripravených zvládnuť potrebnú úroveň programovania.

- Zaujimavy, uzitocny aj potrebny.
- potrebný a užitočný

- Prednášky zaujímavé, prednášajúci je profesionál v danej oblasti, vie inšpirovať študentov k lepším výsledkom zážívnou formou aj prostredníctvom online výučby.

Celkom sa mi pozdával systém priebežného hodnotenia - rozpracovanie semestrálnych prác. Aspoň to človeka nútilo sa tomu priebežne venovať.

Skúsenosti

Cvičenia boli zážívné a užitočné, dalo mi to niečo do života.
Obsah predmetu by bol tazko zvladnutelny bez vykladu z prednasok.
Zúčastnil som sa asi všetkých.

Predmet hodnotím ako jeden z najlepších predmetov na odbore IS. Poznatky z neho tak skoro nezabudnem. Taktiež pán Ing. Jankovič PhD. mal veľmi dobrý prístup ku študentom a snažil sa, aby sme predmet zvládli čo najlepšie, pričom nám bol oporou v prípade potreby v každom čase. Za mňa 11/10.

Je to vyborny predmet ktory vam povie ci fakt ste programator , je logicky a ucivo je taktiez logicke ... ak by som mohol navrhnut zlepsenie tak len take ze by som pridal do vyucovania technologie ktore sa realne v dnesnom svete vyuzivaju a su postavene na stromoch napr elastic search ... atd a dalsi napad je ze na cviceniach

Podľa môjho názoru bolo predstavených pomerne veľa úš, možno by som vyradil zo 2,3. Aj keď na druhú stranu - otázkou je ktoré; takto mám aspoň širší prehľad. Vo všeobecnosti mám z predmetu ale taký "dobrý pocit".

Ing. Peter Jankovič, PhD.

Ďakujem za pozornosť

Využité zdroje: https://knihy.nic.cz/files/edice/Pruvodce_labyrintem_algoritmu.pdf
Prednášky predmetu „Algoritmy a údajové štruktúry 1“
Výsledky z aplikačných príkladov vznikli v spolupráci s: Samuel Kučera, Anton Gašperák.