Algorithms *(IAL)*

Na této stránce je můj elektronický sešit do předmětu IAL. Při jeho vytváření jsem čerpal informace z přednášek a ze studijní opory k předmětu IAL od Prof. Ing. Jana M Honzíka, CSc. Studijní materiály k tomuto předmětu byly dostupné zde.

Tématické okruhy

- Přehled základních datových struktur a jejich použití.
- Principy dynamického přidělování paměti.
- Specifikace abstraktních datových typů (ADT).
- Specifikace a implementace ADT: seznamy, zásobník, fronta, množina, pole, vyhledávací tabulka, graf, binární strom.
- Základní operace nad uvedenými ADT a jejich využití.
- Zásobník a vyčíslování výrazů.
- Algoritmy nad binárním stromem.
- Vyhledávání: sekvenční, v neseřazeném a seřazeném poli, s adaptivní rekonfigurací, vyhledávání se zarážkou, binární vyhledávání, binární vyhledávací strom, vyvážený strom (AVL).
- Vyhledávání v tabulkách s rozptýlenými položkami.
- Řazení, principy, stabilita, řazení bez přesunu položek, řazení podle více klíčů.
- Nejznámější metody řazení a jejich principy: Select-sort, Bubble-sort, Heap-sort, Insert-sort a jeho varianty, Shell-sort, Quick sort v rekurzívní a nerekurzívní notaci, Merge-sort, List-merge-sort, Radix-sort.
- Sekvenční metody řazení.
- Rekurze a algoritmy s návratem.
- Vyhledávání podřetězců v textu.
- Základní principy dokazování programů, antecendence, konsekvence, invariant cyklu, tvorba dokázaných programů.

Algoritmický jazyk

Pseudopříkaz

Pokyn k akci, která se provádí v době překladu. (deklarace, ...)

Příkaz

Pokyn k akci, která se provádí za běhu programu.

Datové typy a struktury

- Jednoduché datové typy
- Strukturované datové typy
- Typ ukazatel

Ordinální typ

Pro každou jeho hodnotu (kromě krajních hodnot) existuje právě jedna následující a právě jedna předchozí hodnota.

Zkrácené vyhodnocování booleovských výrazů (zkratové)

Matematická čistota výrazu je porušena tím, že připouští nedefinovanost částí výrazu. Z praktického pohledu programátora však vede k přehlednějšímu zápisu programu.

Skalární typ

Skalární typ

Jednoduchý typ, pro jehož každé dva prvky lze stanovit, zda jsou si rovny nebo zda je jeden z nich menší nebo větší než druhý.

Strukturovaný typ

Strukturovaný typ se skládá z komponent jiného (dříve definovaného) typu, kterému říkáme **kompoziční typ**.

Pokud jsou všechny komponenty strukturovaného typu stejného typu, nazveme strukturovaný typ **homogenní**.

Komponentám homogenního typu se říká **položky**, zatímco komponentám heterogenního typu se říká **složky**.

Strukturovaný typ má **strukturovanou hodnotu**. Tato hodnota je definovaná, jsou-li definované hodnoty jejích komponent.

Pole

Pole je homogenní **ortogonální** (pravoúhlý) datový typ. Jednorozměrnému poli říkáme **vektor**, dvojrozměrnému poli říkáme **matice**.

Typ pole je specifikován rozsahem svých dimenzí a komponentním typem.

Záznam ***(record)***

Záznam je obecně strukturovaný heterogenní datový typ. Jeho komponenty mohou být libovolného, dříve definovaného typu, říkáme jim složky.

Řídící struktury

Rekurzivní volání funkce

Rekurzívní volání funkce je konstrukce, při níž je funkce volána v těle sebe samé.

Kapitola 7 - Rekurze

Side effect = vedlejší efekt

Změna hodnoty "globální" proměnné uvnitř těla procedury. Častým případem je vstupní parametr předávaný odkazem.

Složitost algoritmů

- Časová složitost
- Paměťová složitost

Čas i prostor potřebný pro algoritmus závisí na množství zpracovávaných dat. Proto má složitost nejčastěji podobu **funkce velikosti dat**, udávané počtem položek `N`.

Asymptotická složitost

Porovnání s funkcí pro `N` blížícímu se nekonečnu.

Omikron ***(velké O)***

Vyjadřuje horní hranici chování.

Omega

Vyjadřuje dolní hranici chování.

Theta

Vyjadřuje **třídu chování**.

Klasifikace algoritmů

Konstantní složitost	`Theta(1)`
Logaritmická složitost	`Theta(log(n))`
Lineární složitost	`Theta(n)`

Linearitmická složitost	`Theta(n * log(n))`
Kvadratická složitost	`Theta(n^2)`
Kubická složitost	`Theta(n^3)`
Exponenciální složitost	`Theta(k^n)`
Binomická časová složitost	`Theta(2^n)`

3. Abstraktní datové typy

Datová struktura

Datová struktura je abstraktní vyjádření zúčastněných vlastností objektů řešeného systému.

Statické přidělování paměti

Staticky se paměť přiděluje deklarovaným datovým strukturám v době překladu. Paměťové úseky jsou pak přístupné pomocí **jména proměnné**, uvedeného v deklaraci.

Dynamické přidělování paměti

Dynamicky přiděluje paměť systém na základě požadavku vzniklého v době řešení programu. K paměťovému úseku se pak přistupuje nepřímo, prostřednictvím **ukazatele**.

Principy dynamického přidělování paměti

Bez regenerace

Se zjednodušenou regenerací

Operace **`mark`** zaznamená stav ukazatele na začátek volné paměti.

Operace **`release`** nastaví ukazatel zpět na stav zaznamenaný operací `mark`.

S regenerací

Nejjednodušší implementace používá zřetězený **seznam volných prvků**. Pomocný ukazatel ukazuje na první volný prvek. Ukazatel posledního prvku je nastaven na `null`.

Abstraktní datový typ

Abstraktní datový typ (ADT) je definován množinou hodnot, které může nabývat a množinou operací nad ním definovaných.

Abstraktní datový typ representuje třídu prvků se shodnými vlastnostmi. Jednomu konkrétnímu prvku říkáme:

- proměnná daného typu
- prvek daného typu
- **instance daného typu**
- výskyt daného typu
- abstraktní datová struktura daného typu

ADT umožňuje manipulaci s daty bez znalosti jejich vnitřní reprezentace.

Generický ADT

Generický ADT je definovaný **pouze množinou operací** nad ním definovaných. Používá se zpravidla k tvoření **konkrétních ADT**.

Konstruktor

Konstruktor je operace, jejímž vstupním parametrem je výčet všech komponent

struktury a výsledkem operace je datová struktura obsahující tyto komponenty.

Selektor

Selektor je operace, která umožní přístup ke komponentě datové struktury na základě uvedení **jména struktury** a zápisu **reference**.

Iterátor

Iterátor je operace, která umožňuje provádět zadanou činnost nad všemi prvky **homogenní** datové struktury.

Destruktor

Destruktor je operace nad dynamickou strukturou, která ji zruší a vrátí prostor jí zaujímaný.

Diagram signatury

Diagram signatury se používá ke znázornění **syntaxe ADT**.

Každý typ je zobrazen **oválem**. Každá operace je zobrazena **malým kroužkem**. **Šipky** znázorňují vstupní a výstupní parametry operace. Několik operací se stejnou syntaxí může být reprezentováno jedním kroužkem.

Sémantika ADT

- **slovně**
- **axiomaticky** - pomocí množiny axiomů
- **operační specifikace** - pomocí funkcí popisujících chování operace. Nevýhoda spočívá v tom, že tato specifikace vyjadřuje konkrétní **implementaci** ADT.

Seznam

Seznam je homogenní lineární dynamická struktura.

Lineární znamená, že každý prvek (s výjimkou krajních) má právě jednoho následníka a právě jednoho předchůdce.

Rozdělení podle přístupu k prvkům seznamu:

- **Jednosměrně vázaný** seznam (jednosměrný)
- **Dvousměrně vázaný** seznam (dvousměrný)
- **Kruhový** seznam

Dvousměrný seznam

Dvousměrný seznam je stranově symetrická datová struktura.

Lexikografická relace

Lexikografická relace dvou lineárních struktur porovnává odpovídající dvojice prvků zleva. V případě nerovnosti určuje výsledek relace neshodná dvojice. Pokud jedna struktura končí dříve než druhá, pak je menší.

Ekvivalence dvou seznamů *(rekurzivní definice)*

Dva seznamy jsou ekvivalentní, když jsou oba prázdné nebo když se rovnají jejich první prvky a současně jejich zbývající části.

Hlavička

Hlavička je první prvek seznamu, který není skutečným prvkem, ale má pomocnou funkci. **Prázdný seznam** pak obsahuje jediný prvek - hlavičku.

Kruhový seznam

Lze vytvořit nastavením ukazatele na následující prvek posledního prvku na prvek první. Kruhový seznam tak ze sémantického pohledu nemá začátek ani konec. Může být opět jednosměrně i dvousměrně vázaný.

Soubor operací je nutno doplnit o **průchod celým seznamem**. To lze implementovat zavedením počítadla prvků a následným použitím počítaného cyklu.

Zásobník

Opět homogenní lineární dynamická struktura. Lze odvodit z lineárního seznamu **omezením přístupu** k jednomu jeho konci.

Zásobník lze využít k:

- inverzi pořadí lineárního uspořádání
- konstrukci rekurzivních algoritmů
- zpracování výrazů - **infixová**, **prefixová** a **postfixová notace**
- algoritmy s návratem *(back-tracking)*
- předávání parametrů
- ...

Fronta

Opět homogenní lineární dynamická struktura. Na jednom konci fronty (obvykle konec) se vkládají nové prvky. Na druhém konci (obvykle začátek) se prvky odebírají.

Fronta implementovaná polem

obdoba kruhového seznamu v poli

Oboustranně ukončená fronta *(DEQUE = Double Ended QUEue)*

Oboustranně ukončená fronta umožňuje přidávat i odebírat prvky na obou koncích.

Vyhledávací tabulka *(look-up table)*

Vyhledávací tabulka je homogenní, **obecně dynamická** struktura.

Každá položka tabulky obsahuje složku, které se říká **klíč**. Klíč má v tabulce jednoznačnou hodnotu, která slouží k identifikaci/vyhledání položky.

Tabulka má vlastnosti **kartotéky**. Operace vyhledávání *(look-up)* patří mezi nejzákladnější operace implementované nad tabulkou.

Pole

Mapovací funkce

Mapovací funkce je vztah, který převádí *n*-tici indexů prvku *n*-dimenzionálního pole na index jednorozměrného pole, jímž je *n*-dimenzionální pole reprezentováno.

Době výpočtu mapovací funkce se říká **přístupová doba***(access time)*. Paměťově úspornější implementace pole mají zpravidla delší přístupovou dobu.

Matice s nestejně dlouhými řádky

Řídké pole

Řídké pole je pole v němž má významné množství prvků stejnou hodnotu, této hodnotě říkáme **dominantní**.

Uchovávají se pouze hodnoty nedominantních prvků spolu s jejich indexy. Přístup se pak realizuje **vyhledáváním**, například pomocí vyhledávací tabulky.

Graf

Graf je definován trojicí $G = (N, E, I)$, kde

- N je **množina uzlů**, jimž lze přiřadit hodnotu.
- E je **množina hran**, kterým lze přiradit hodnotu. Každá hrana spojuje dva uzly a může být **orientovaná**. Jsou-li hrany orientované, grafu se pak říká **orientovaný graf**.
- I je **množina spojení**, která určuje spojení dvojic uzlů daného grafu.

Průchod je posloupnost všech uzlů grafu.

Průchod je operace nad grafem, která provádí transformaci z nelineární struktury na lineární.

Cesta z uzlu `A` do uzlu `B` je posloupnost hran, po nichž se dostaneme z uzlu `A` do uzlu `B`, aniž bychom šli po nějaké hraně dvakrát.

Cyklus je neprázdná cesta, která končí v témže uzlu v němž začíná. Graf, který obsahuje cyklus, se nazývá **cyklický graf**. Graf, který neobsahuje cyklus se nazývá **acyklický graf**.

Strom

Kořenový strom ***(root tree)***

Kořenový strom je acyklický orientovaný graf, který má jeden zvláštní uzel. Tento uzel se nazývá kořen *(root)*.

Kořen je uzel pro nějž platí, že z každého uzlu stromu vede jen jedna cesta do kořene.

Z každého uzlu (kromě kořene) vede právě jedna hrana směrem ke kořeni do uzlu, který se nazývá **otcovský**. Z každého uzlu vede libovolný počet hran k uzlům, kterým se říká **synovské**.

Výška prázdného stromu je 0, výška stromu s jediným uzlem je 1. V jiném případě je výška stromu dána počtem hran od kořene k nejvzdálenějšímu uzlu **+ 1**.

Binární strom ***(rekurzivní definice)***

Binární strom je buď prázdný, nebo sestává z jednoho uzlu zvaného kořen a dvou podstromů - levého a pravého.

Neterminální uzel má ukazatel na jeden nebo dva synovské uzly. Uzel, který nemá žádné synovské uzly, se nazývá **terminální**.

Binární strom je **váhově vyvážený**, když pro všechny jeho uzly platí, že počty uzlů jejich levého podstromu a pravého podstromu se rovnají nebo se liší právě o 1.

Neprázdnému binárnímu stromu o výšce *n*, který obsahuje právě 2^n-1 uzlů, říkáme **absolutně váhově vyvážený**.

Binární strom je **výškově vyvážený**, když pro všechny jeho uzly platí, že výška levého podstromu se rovná výšce pravého podstromu nebo se liší právě o 1.

Průchod stromem

PreOrder	1.**this**	2.**LPtr**	3.**RPtr**
InOrder	1.**LPtr**	2.**this**	3.**RPtr**
PostOrder	1.**LPtr**	2.**RPtr**	3.**this**
InvPreOrder	1.**this**	2.**RPtr**	3.**LPtr**
InvInOrder	1.**RPtr**	2.**this**	3.**LPtr**
InvPostOrder	1.**RPtr**	2.**LPtr**	3.**this**

Průchod `PostOrder` je *obrácený* průchod `InvPreOrder`.

4. Vyhledávání

- sekvenční vyhledávání - v seřazeném/neseřazeném poli
- nesekvenční vyhledávání v poli
- binární vyhledávací strom
- tabulka s rozptýlenými položkami

Přístupová doba ***(access time)***

Přístupová doba je doba potřebná k vyhledání položky s hledaným klíčem. Uvádí se maximální, minimální a průměrná doba úspěšného a neúspěšného vyhledání (6 kombinací).

Vyhledávací klíč

Při výběru vyhledávacího algoritmu je potřeba znát vlastnosti vyhledávacího klíče. Rozlišujeme, je-li nad klíčem definována pouze relace ekvivalence nebo i relace uspořádání.

Klíč může být jednoduchý nebo strukturovaný. Pro ekvivalenci dvou strukturovaných klíčů musí být ekvivalentní všechny odpovídající si složky klíče.

Typy tabulek podle dynamiky

- **Statická** - seznam obcí v republice
- **Statická po etapách** - tel. seznam aktualizovaný po roce
- **Dynamická s ohledem na vkládání**
- **Plně dynamická** - dovoluje rušení položek

Zaslepení

Zalepení je přepsání klíče rušené položky takovou hodnotou klíče, o níž je jisté, že nebude nikdy vyhledávána.

Sekvenční vyhledávání

Sekvenční vyhledávání lze snadno realizovat v typicky sekvenčních strukturách, jako je seznam nebo soubor.

Sekvenční vyhledávání

Min. čas úspěšného vyhledání: `1`

Max. čas úspěšného vyhledání: `N`

Avg. čas úspěšného vyhledání: `N/2`

Čas neúspěšného vyhledání: `N`

Sekvenční vyhledávání se zarážkou

Sekvenční vyhledávání v seřazeném poli

Urychlí se pouze čas neúspěšného vyhledání. Opět existuje varianta se zarážkou.

Sekvenční vyhledávání s adaptivní rekonfigurací

Po každém úspěšném vyhledání položky se položka vymění se svým levým sousedem, pokud již sama není na první pozici.

Binární vyhledávání v seřazeném poli

Binární vyhledávání se provádí nad seřazenou množinou klíčů. Je vyžadován náhodný přístup k položkám. Metoda je založena na **půlení intervalu**.

Max. čas úspěšného/neúspěšného vyhledání: $\log_2(N)$

Binární vyhledávací strom

Víme...

Dijkstrova varianta binárního vyhledávání

Dijkstrova varianta se používá pro účely řazení. Předpokládá více položek se stejným klíčem. Algoritmus nekončí nalezením shody s klíčem, ale nalezením **posledního prvku** v posloupnosti prvků se stejným klíčem.

Uniformní a Fibonacciho vyhledávání

Uniformní a Fibonacciho vyhledávání je vhodné v případě, že je operace půlení intervalu časově náročná operace. Tento případ nastává, pokud jsou čísla kódována jinak než v binárním kódu (např. v BCD).

Uniformní binární vyhledávání

Uniformní binární vyhledávání je založeno na práci s **odchylkou od středu**. Tyto odchylky jsou na dané úrovni vyhledávacího stromu vždy stejné, proto se vyhledávání jmenuje uniformní. Odchylky pro jednotlivé úrovně jsou dopředu spočítány a uloženy v poli odchylek.

Tabulky, které mají počet prvků o hodnotě 2^n-1, mají pole odchylek se společným základem. Pro takové tabulky lze připravit **univerzální pole odchylek**. Pro tabulky s libovolnou velikostí se používá Sharova metoda.

Fibonacciho binární vyhledávání

Místo binárního rozhodovacího stromu je použit **Fibonacciho strom**. Základem Fibonacciho stromu je Fibonacciho posloupnost 1. řádu.

Sharova metoda

Sharova metoda řeší případ, kdy skutečný počet prvků tabulky je nevhodný pro Uniformní nebo Fibbonacciho vyhledávání.

Tabulka je rozdělena na největším indexu, který vyhovuje metodě a je menší než daná velikost. Pokud je hledaný klíč v tomto intervalu, postupuje se jako obvykle. Pokud je hledaný klíč mimo tento interval, provede transformaci pole doprava tak, aby prohledávaná část tabulky měla opět vhodný počet prvků.

Binární vyhledávací strom (BVS)

Uspořádaný strom

Uspořádaný strom je kořenový strom, pro jehož každý uzel platí, že n-tice kořenů podstromů uzlu je uspořádaná.

Binární vyhledávací strom

Binární vyhledávací strom je uspořádaný binární strom, pro jehož každý uzel platí, že klíče všech uzlů levého podstromu jsou menší než klíč v uzlu a klíče všech uzlů pravého podstromu jsou větší než klíč v uzlu.

(rekurzivní definice) Binární vyhledávací strom je buď prázdný nebo sestává z kořene, jehož hodnota klíče je větší než hodnota všech klíčů levého binárního vyhledávacího podstromu a je menší než hodnota všech klíčů pravého binárního podstromu.

Průchod `InOrder` binárním vyhledávacím stromem dává posloupnost prvků seřazenou podle velikosti klíče.

Vyhledávání v BVS je podobné binárnímu vyhledávání v seřazeném poli.

Operace vkládání uzlu

Operace vkládání má aktualizační sémantiku - pokud uzel s daným klíčem existuje, přepíše se jeho hodnota hodnotou novou. Pokud neexistuje, vloží operace nový uzel jako **terminální uzel** na správné místo v BVS.

Operace rušení uzlu

Terminální uzel Uzel se jednoduše zruší.

S jedním synem Synovský uzel se připojí na nadřazený uzel rušeného uzlu. Potom se samotný uzel zruší.

Se dvěma syny Uzel se neruší fyzicky, ale jeho hodnota se přepíše hodnotou jiného uzlu, který lze zrušit snadno. Takovým uzlem je:

- nejpravější uzel levého podstromu
- nejlevější uzel pravého podstromu

Binární vyhledávací strom se zarážkou

(viz. vyhledávání v poli se zarážkou) Zarážka je speciální uzel, na který se odkazují všechny terminální uzly. Do tohoto uzlu se vloží hledaný klíč, nemusí se potom ověřovat, jestli existuje synovský uzel.

Binární vyhledávací strom se zpětnými ukazateli

Při průchodu `Inorder` není potřeba používat zásobník. Zpětný ukazatel se nastavuje podle následujících pravidel:

Kořen Zpětný ukazatel je `NULL`.

Levý syn Zpětný ukazatel ukazuje na svého otce.

Pravý syn Zpětný ukazatel se dědí od otce.

Vyvážený strom (AVL)

AVL strom

AVL strom je **výškově vyvážený**. Lze dokázat, že jeho výška je maximálně o 45% vyšší než výška váhově vyváženého. Horní hranice složitosti *(omikron)* vyhledávání je tedy $1.45\ log_2 N$, pro strom o N uzlech.

Znovuustavení výškové vyváženosti AVL stromu lze provést **rekonfigurací** uzlů v okolí kritického uzlu. **Kritický uzel** je nejvzdálenější uzel od kořene, v jehož podstromu je porušena rovnováha.

Váha uzlu

Váha se ukládá do každého uzlu AVL stromu za účelem udržování jeho výškové vyváženosti. Váha uzlu se označuje písmenem `h` a nabývá následujících hodnot:

`h = 0` Uzel je absolutně výškově vyvážený.

`h = -1` Uzel je těžký vlevo.

`h = 1` Uzel je těžký vpravo.

`h = -2` Uzel má porušenou výškovou vyváženost doleva.

`h = 2` Uzel má porušenou výškovou vyváženost doprava.

Rotace

Mechanismu znovuustavení právě porušené výškové vyváženosti se říká rotace. Rotace se dělí na jednoduché (LL, RR) a dvojité (DLR, DRL).

Tabulky s rozptýlenými položkami (hash table)

Tabulka s přímým přístupem

Existuje-li izomorfní funkce f, která zobrazuje každý prvek z množiny klíčů do množiny sousedních míst v paměti, označujeme tabulku jako tabulku s přímým přístupem a funkci f jako mapovací funkci tabulky s přímým přístupem.

Časová složitost přístupu: `1`

Tabulka s nepřímým přístupem

Jevu, kdy se dva různé klíče zobrazí do stejného místa v tabulce, říkáme **kolize**. Dvěma nebo více klíčům, které se namapují do téhož místa říkáme **synonyma**.

Mapovací funkce

Mapovací funkce mapuje klíč na index do rozptylového (hashovacího) pole. Dobrá mapovací funkce produkuje co nejméně kolizí a její výpočet je rychlý.

Seznam synonym

Explicitně zřetězený Každý prvek seznamu obsahuje adresu následníka.

Implicitně zřetězený Adresa následníka je funkcí adresy předchůdce.

Implicitní zřetězený seznam synonym s pevným krokem

Seznam synonym je tvořen po sobě jdoucími prvky a je ukončen první volnou položkou pole. S polem se pracuje jako s kruhovým seznamem. Tabulka musí obsahovat aspoň jeden volný prvek. Velikost tabulky by měla být prvočíslo.

Metoda s dvojí hashovací funkcí

Krok seznamu synonym je určen druhou hashovací funkcí.

5. Řazení

Třídění

Třídění *(sorting)* je rozdělování položek do skupin (tříd) podle společných vlastností (atributů).

Řazení

Řazení *(ordering)* je uspořádávání položek do posloupnosti podle relace uspořádání nad zvoleným klíčem.

Setřídění

Setřídění *(merging)* je sloučení dvou nebo více seřazených lineárních homogenních datových struktur do jedné seřazené lineární homogenní datové struktury.

Vlastnosti řadících algoritmů

Stabilita

Stabilita řazení je vlastnost algoritmu, který zachová relativní pořadí položek se stejnou hodnotou klíče.

Přirozenost

Pro přirozený řadící algoritmus platí, že doba řazení již uspořádané posloupnosti je menší, než doba řazení náhodně uspořádané posloupnosti a ta je menší, než doba řazení opačně seřazené posloupnosti.

Řazení podle více klíčů

Složená relace

Metoda spočívá ve vytvoření takové relace uspořádání, která zohlední všechny klíče.

Postupné řazení podle jednoho klíče

Postupuje se od klíče s nejnižší prioritou. Podmínkou je použití **stabilní** řadící metody.

Algomerovaný klíč

Uspořádaná n-tice klíčů se konvertuje na vhodný typ (často řetězec), nad nímž je definována relace uspořádání.

Řazení bez přesunu položek

Jednou z nejčastějších operací prováděných během řazení je přesun položek. Pokud jsou přesouvané položky objemné, je tato operace časově náročná. Při použití řazení bez přesunu položek se přesouvají pouze odkazy na položky.

MacLarenův algoritmus

MacLarenův algoritmus uspořádá pole seřazené bez přesunu položek na místě samém, tj. bez pomocného pole.

Algoritmus pracuje s původním polem, jehož položky jsou zřetězené do seřazeného seznamu. Umístí se první položka na své místo a upraví se ukazatel v položce, který

na ni ukazoval. Stejným způsobem se přesune druhá položka na své místo... Takhle se projde celým seznamem.

Klasifikace řadících algoritmů

Podle přístupu k paměti

- sekvenční přístup
- náhodný přístup

Podle typu procesoru

- sériové
- paralelní

Podle principu řazení

Princip výběru *(selection)*

Přesouvají minimum/maximum do výstupní posloupnosti.

Princip vkládání *(insertion)*

Vkládají postupně prvky do seřazené výstupní posloupnosti.

Princip rozdělování *(partition)*

Rozdělují postupně množinu prvků na dvě podmnožiny tak, že prvky jedné jsou menší než prvky druhé.

Princip slučování *(merge)*

Setřiďují postupně seřazené dvě podmnožiny do jedné.

Select Sort

Select Sort je řazení na principu výběru.

Jádrem algoritmu je cyklus pro vyhledání pozice extrémního (min. nebo max.) prvku v zadaném segmentu pole. Nalezený prvek se vždy vymění s prvkem pole na odpovídajícím indexu.

Metoda je nestabilní, přirozená(?) a má kvadratickou časovou složitost.

Bubble Sort

Bubble Sort je řazení na principu výběru.

Víme...

Metoda je stabilní, přirozená a má kvadratickou časovou složitost. Nejrychlejší metoda v případě již seřazené posloupnosti!

Ripple-sort

Ripple-sort si pamatuje polohu první výměny a díky tomu přeskočí dvojice, u nichž je jasné, že se nebudou vyměňovat.

Shaker-sort

Shaker-sort střídá směr probublávání (houpačková metoda).

Heap sort

Hromada

Hromada *(heap)* je struktura typu strom. Pro všechny její uzly platí, že mezi

otcovským uzlem a všemi jeho synovskými uzly je stejná relace uspořádání. Její často používaná varianta je **binární hromada**, která je založena na binárním stromu.

Hromada je vhodná tam, kde je potřeba opakovaně hledat extrémní prvky na téže množině prvků.

Operace `sift` *(prosetí, zatřesení)*

Operace `sift` provede rekonstrukci hromady, jejíž rovnováha byla porušena v kořeni. Tato operace má v nejhorším případě složitost $log_2 N$ pro hromadu o N uzlech.

Algoritmus Heap-sortu

Binární hromada je implementována polem pomocí **implicitního zřetězení**. Je-li otcovský uzel na indexu `i`, potom jeho levý syn je na indexu `2i` a jeho pravý syn na indexu `2i+1`.

1. Ustaví se se hromada tím, že se postupně zatřese všemi otcovskými uzly. Začíná se od pravého dolního a postupuje se směrem doleva a nahoru.
2. Potom se v cyklu prochází polem od konce. Vždy se vymění první prvek (kořen hromady) s největším prvkem v poli. Největší prvek v poli se najde pomocí zatřesení hromadou, tj. s logaritmickou složitostí.

Vlastnosti Heap-sortu

- Složitost: $N * log_2 N$ (linearitmická)
- **nestabilní**
- **nepřirozená**

Insert sort

Pole je rozděleno na dvě části - levou seřazenou a pravou neseřazenou. Na začátku tvoří levou část první prvek. V cyklu se berou prvky z neseřazené části a vkládají se do seřazené části na správné místo. Zbytek seřazené části se musí vždy o 1 posunout (vkládání do pole, víme...).

Bubble-insert sort

Bubble-insert sort *(metoda bublinkového vkládání)* slučuje vyhledání místa pro vložení a posun segmentu pole do jednoho cyklu postupným porovnáváním a výměnou dvojic prvků.

Metoda je **stabilní**, **přirozená** a má **kvadratickou** časovou složitost.

Vkládání s binárním vyhledáváním

Metoda pracuje stejně jako Insert-sort, ale místo sekvenčního vyhledávání používá binární vyhledávání. Pokud má být metoda stabilní, je třeba použít Dijkstrovu variantu binárního vyhledávání.

Složitost vyhledávání se tak sníží z lineární na logaritmickou. Počet přesunů je však stále vysoký a metoda proto nepřinese výrazné zrychlení.

Quick sort

[Ukázka]

Quick sort je založen na mechanismu rozdělení *(partition)*.

1. Vybere hodnotu ze středu intervalu - **pseudomedián**.
2. Najde první hodnotu zleva, která je větší než pseudomedián.
3. Najde první hodnotu zprava, která je menší než pseudomedián.
4. Tyto hodnoty vymění a vrátí se na bod 2, dokud se indexy nepřekříží.
5. Rekurzivně zavolá funkci QuickSort pro zbývající 2 části.

Nerekurzivní zápis

Nerekurzivní zápis Quick-sortu používá explicitně definovaný zásobník. Zásobník musí být dimenzovaný pro `N-1` dvojic indexů (nejhorší případ).

Vlastnosti Quick sortu

- Složitost: $N * \log_2 N$ (linearitmická)
- nestabilní
- nepřirozená

Shell sort

Shell sort *(řazení se snižujícím se přírustkem)* je metoda založená na principu bublinkového vkládání. V několika průchodech se řadí prvky stejně vzdálené od sebe pomocí bublinkového vkládání. Vzdálenost prvků od sebe *(krok)* se pak postupně snižuje, až je roven jedné, tj. klasický Bubble-insert.

Teoretické analýzy nenašly nejvhodnější řadu snižujících se kroků. Kerninghan a Ritchie ve své implementaci používali krok `N/2`, který pak postupně dělili dvěma.

Shell sort je nestabilní řadící metoda. Její experimentálně naměřená složitost odpovídá linearitmickým řadícím metodám. Na rozdíl od HeapSortu nepotřebuje na začátku ustavit hromadu. A narozdíl od QuickSortu nepotřebuje ke své činnosti zásobník.

Merge sort

Merge sort pracuje na principu setřiďování posloupností. Ke své činnosti vyžaduje dvojnásobnou velikost pole. Postupuje zdrojovým polem zleva a současně zprava a setřiďuje dvě (proti sobě postupující) neklesající posloupnosti. Výsledek se ukládá do cílového pole. Potom si pole vymění role - zdrojové se používá jako cílové a naopak - tzv. houpačkový mechanismus. Algoritmus končí, vznikne-li jedna seřazená posloupnost.

Vlastnosti

- Složitost: $N * \log_2 N$ (linearitmická)
- nestabilní
- nepřirozená

List Merge Sort

List Merge Sort pracuje na principu setřiďování posloupností, přičemž využívá řazení bez přesunu položek.

V prvním kroku se zřetězí všechny neklesající posloupnosti, jejich začátky se přitom uloží do pomocného seznamu. Potom se v každém kroku setřídí dvě seřazené posloupnosti do jedné. Algoritmus končí, když je v seznamu jen jedna seřazená posloupnost. Výsledek je potom možné přeskládat v poli pomocí MacLarenova algoritmu.

Existují stabilní varianty této řadící metody. Experimentálně naměřená časová složitost odpovídá linearitmickým řadícím metodám.

Radix sort

Metoda je založena na principu třídění. Pracuje bez přesunu položek, každá položka proto obsahuje ukazatel na následující položku. Kromě toho je potřeba uchovávat seznamy na začátky a konce již seřazených seznamů podle jednotlivých řádů.

Řadí se postupně od cifry s nejnižším významem.

Radix sort je stabilní řadící metoda. Stav uspořádání nemá podstatný vliv na čas a proto se chová nepřirozeně. Teoretická časová složitost je lineární :-)

Sekvenční řazení

Třípásková přímá metoda

Každý průchod souborem má dvě fáze - distribuční a setřiďovací. Během **distribuční** fáze se prvky zdrojového souboru rovnoměrně rozdělí do dvou souborů. V **setřiďovací** fázi se potom setřiďují posloupnosti z obou souborů a výsledek se ukládá do (původně) zdrojového souboru.

Počet průchodů je $log_2 N$, kde N je počet prvků seřazované posloupnosti.

Třípásková přirozená metoda

Na rozdíl od přímé metody, přirozená metoda setřiďuje neklesající posloupnosti zdrojových souborů. Počet průchodů je $log_2 N$, kde N je počet **neklesajících posloupností** zdrojového souboru.

Čtyřpásková přirozená metoda

Setřiďovaná posloupnost není ukládána zpět na zdrojovou pásku, ale je rovnou distribuována na další dvě pásky. Celkem jsou proto potřeba 4 pásky (nepočítají se zdrojová data).

Mnohacestné vyvážené setřiďování

Mnohacestné vyvážené setřiďování je zobecněním čtyřpáskové přirozené metody. Používá $2k$ pásek, kde $k>2$. Místo dvou posloupností se v každém průchodu setřiďuje k posloupností. Počet průchodů je $log_k N$.

Polyfázové seřazování

Polyfázové seřazování je založené na **Fibonacciho posloupnosti**. Dosahuje stejné rychlosti jako mnohacestné vyvážené setřiďování, ale pracuje pouze s $k + 1$ soubory.

6. Vyhledávání v textu

Naivní algoritmus

Víme... Horní hranice složitosti je $O(m*n)$, kde m je délka vzorku a n je délka prohledávaného textu. V přirozených jazycích jsou ale mezní případy nepravděpodobné.

KMP algoritmus

(Knuth Morris Prattův) Ke své práci využívá **konečný automat**. Z každého uzlu vychází tolik hran, kolik je znaků abecedy. Automat se po načtení jednoho znaku posune na další stav, pokud je shoda, jinak se vrátí na některý předchozí stav.

KMP algoritmus provede maximálně $2n$ porovnání a horní mez jeho složitosti je $O(m+n)$. Při zpracování přirozeného jazyka dosahuje stejné rychlosti jako předchozí algoritmus, ale na rozdíl od něj nevyžaduje návrat v řetězci.

BMA algoritmus

BMA *(Boyer Moorův)* algoritmus přeskakuje porovnávání znaků, o nichž lze prohlásit, že nemohou odpovídat vzorku. Čím delší je hledaný vzorek, tím více znaků lze přeskočit. BMA používá dvě heuristiky.

První heuristika

Položíme začátky vedle sebe a porovnáme poslední znak. Pokud se shoduje, porovnává se vzorek odzadu. Pokud se neshoduje, můžeme se posunout dále - nutno spočítat o kolik!!

Druhá heuristika

Druhá heuristika se uplatní, pokud se ve vzorku vyskytuje nějaký podřetězec 2x. Jestliže se nalezne shoda v podřetězci, ale dál už ne, posune se vzorek tak, aby nalezený kus textu odpovídal druhému podřetězci.

Vlastnosti BMA

Rychlost algoritmu BMA závisí na kardinalitě abecedy a opakování podřetězců ve vzorku. Měřením bylo zjištěno, že pro délku vzorku větší než 5 se provádí asi **0.3 porovnání z počtu znaků** v prohledávaném textu.

7. Rekurze

Rekurze umožňuje definovat nekonečnou množinu objektů konečným popisem. Rozlišujeme rekurzi v datových a řídících strukturách - víme...

Efektivní vyčíslitelnost

Funkce $f(x_1, x_2, \ldots, x_n)$ je **efektivně vyčíslitelná**, existuje-li formální postup, který dokáže stanovit hodnotu funkce pokaždé, jsou-li zadány relevantní parametry $x_1, x_2, \ldots, x_n$. Takový postup nazýváme **algoritmem**.

Rekurzivní vyčíslitelnost

Churchova teze říká, že jakoukoli intuitivně vyčíslitelnou funkci lze popsat pomocí rekurzivních funkcí. Programování založené na rekurzivních funkcích označujeme jako **funkcionální programování**.

Rekurze a iterace

Řešení úloh pomocí rekurze je založeno na principu *divide and conquer*. Úloha je rozložena na podúlohy, které se řeší podobným způsobem - **redukce**. Po konečném počtu redukcí vzniknou úlohy, které lze řešit přímo.

Lze dokázat, že rekurze a iterace mají **stejnou vyjadřovací schopnost**. Díky tomu je možné rekurzivně definované algoritmy zapsat iterací - to se používá zejména pro zvýšení rychlosti a snížení paměťových nároků algoritmu.

Konečnost rekurze

Každá rekurze musí někdy skončit, víme...

Využití rekurze

- **Rekurzivně definované problémy**
  (Hanojské věže, Hilbertovy křivky, ...)
- **Algoritmy s návratem** = Back tracking
  (Cesta koně, Osm dam, ...)
- **Funkcionální jazyky** - neznají zápis pro iteraci

8. Dokazování správnosti algoritmu

Základním principem dokazování je stanovení oborů hodnot proměnných a vztahů mezi proměnnými pro každý příkaz programu. Používají se k tomu následující pravidla.

Pravidlo 1

Pro každý příkaz*(S)* programu jsou stanoveny podmínky, které platí před*(P)* a po*(Q)* provedení příkazu. Tvrzení platné před příkazem se nazývá **antecendence***(precondition)*. Tvrzení platné po provedení příkazu se nazývá **konsekvence***(postcondition)*. Zapisuje se jako ***(S,P)=>Q***.

Pravidlo 2

Spojuje-li se před příkazem několik větví algoritmu, pak konsekvence všech předcházejících příkazů musí logicky implikovat antecendenci následujícího příkazu.

Pravidlo 3

Platí-li před podmíněným příkazem s podmínkou *B* tvrzení *P*, pak konsekvence příkazu v podmíněné větvi je ***B && P*** a konsekvence příkazu v alternativní větvi je ***!B && P***.

Pravidlo 4

Je-li *P* antecendence přiřazovacího příkazu ***a=b***, konsekvenci přiřazovacího příkazu dostaneme tak, že každý výskyt výrazu *b* v antecendci *P* nahradíme L-hodnotou *a*.

Je-li *Q* konsekvence přiřazovacího příkazu ***a=b***, antecendenci přiřazovacího příkazu dostaneme tak, že každý výskyt L-hodnoty *a* v konsekvenci *Q* nahradíme výrazem *b*.

Invariantní tvrzení

Invariantní tvrzení vzhledem k příkazu *S* je současně antecendencí i konsekvencí příkazu *S*. To znamená, že tvrzení je neměnné vzhledem k příkazu *S*.

Pravidlo 5

Je-li dáno invariantní tvrzení *P* pro příkaz cyklu `while (B)`, pak konsekvencí cyklu je ***P && !B***.

Dokazování správnosti indukcí

Matematickou indukci lze využít například ke stanovení invariantu cyklu. Matematická indukce vychází z **axiomů o přirozených číslech**:

- Číslo 1 je přirozené číslo.
- Je-li číslo *n* přirozené číslo, pak číslo *n+1* je přirozené číslo.

Dijkstrova tvorba dokázaných programů

Předchozí metody dokazovaly správnost intuitivně zapsaných algoritmů. Dijkstrova tvorba dokázaných algoritmů umožňuje vytvářet úplné a správné algoritmy na základě matematických transformací.