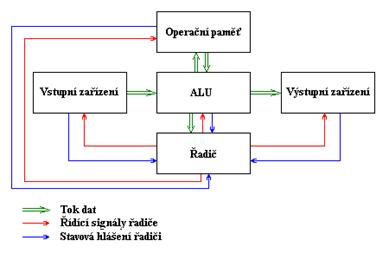
# Vnější a vnitřní paměť.

### Vnítřní paměť

Paměť, ke které má procesor přímý přístup  


von Neumannové schema

Rychlá, libovolný přístup

při vypnutí PC se obsah ztratí

Její správu a alokaci obstarává zpravidla OS

Lze ji dále dělit na:

Operační paměť– zpravidla typu RAM

Cache procesoru

Registry procesoru a chipsetu

### Data ve vnitřní paměti – přístup k datům

Každé paměťové místo v RAM má svou unikátníadresu

Adresace pamětí – viz. předmět Operační systémy

Data lze ukládat a následněk nim přistupovat pomocí výše zmíněných adres

Adresy jsou většinou vyjadřovány pomocíhexadecimálních čísel

Různé datové typy mohou alokovat různěveliký paměťový prostor

Programátor zřídka pracuje přímo s adresami – viz dále Konstantya Proměnné

**Vnější paměť**

Paměť určená k trvalému ukládání dat

Procesor k ní nemá zpravidla přímý přístup

Pomalejší než vnitřní paměť, sekvenční nebo libovolný přístup

Je stálá – při vypnutí PC se obsah neztratí

OS k přístupu používá ovladače zařízení, příp. souborový systém aj.

Data jsou zpravidla organizovány v souborech

Sekvenční přístup

Používají jej magnetické pásky

Primárněurčeno pro zálohování

Nelze okamžitěpřejít na konkrétní data, je nutno číst celou sekvenci od počátku – vysoká latence (zpoždění)

Neexistují adresáře, soubory, žádná organizace

Přímý přístup

Lze okamžitěpřejít na konkrétní data - adresace

HDD, SSD, optické disky, flash, floppy, ZIP …

Data uložená v souborech, hierarchie pomocí adresářů

Přístup k souborům pomocí souborového systému

Data uložena jako textováči binární

Rychlost závisí na optimálním uložení dat a technických parametrech

# Datové typy a jejich rozdělení

Void

Jednoduché datové typy

Jsou definovány identifikátorem a datovým typem v deklarační oblasti

int(typ) vek(identifikátor);

Definovány relace

= > < !=

V paměťovém prostoru uložena přímo hodnota této proměnné/konstanty

Ordinální datové typy - vlastnosti

Hodnota má svého

předchůdce pred(x)

následovníka succ(x)

Její pozici lze číselněohodnotit – ordinálníčíslo ord(x)

Základní ordinální typy

integer – celé číslo (1, 45, 9, -7 ….)

boolean – logická hodnota true/false

char – znak ('w', 't', '8' …)

interval – viz dále

enum – výčtový typ, viz dále

interval

Souvislá neprázdná podmnožina hodnot ordinálního typu

Horní a dolní mez dána konstantou daného ordinálního typu

enum

Datový typ definovaný uživatelem

Jednotlivé hodnoty mají přiřazeny svá ordinálníčísla, počínaje 0

Neordinální datový typ

real

Konečná podmnožina hodnot reálného typu

Reálný typ – hodnoty a pohyblivou desetinnou čárkou

V paměti uložen jako dvojice hodnot (M,N), kde M – mantisa N - exponent

Má omezení – existuje pouze tzv. strojová nula určena intervalem (0,minreal) minreal - absolutní hodnota minimálnězobrazitelné hodnoty na daném HW S touto konečnou přesností se vykonávají i všechny operace s reálnými čísly

Možno provádět konverze integer <==> real – ztráta přesnosti

Strukturované datové typy

Skupina proměnných jednoduššího typu

Skupina jednoduchých typů

Skupina "nižších" strukturovaných typů

Jejich kombinace

Poskytuje prostředky pro práci s jednotlivými prvky

Základní strukturované typy

Pole – kolekce prvků stejného typu

String –řetězec, pole znaků

Záznam (struct)

Množina

Soubor

Dynamické datové struktury

ADT abstraktní datové typy

# Strukturované DT, co víte o polích a řetězcích

Struktorované DT je skupina jednodušších typ

poskytuje prostředky pro práci s prvky dat, tj. pro jejich zpřístupnění

Konkrétní strukturované typy opět závisína programovacím jazyce(string, array)

**Pole**

ma pevný počet prvků,

přistup pomoci indexu,

každý index má pravé jeden přiřazený prvek,

int[] a = new int[20];

index je většinou integer, nebo jiný ordinální typ

nebo řetězec a to bude asociativní pole ADT tabulka, mapa , HashMap

můžou byt dynamické

**Řetězec(String)**

pole znaků

OOP jazyky to maji jako objektový typ

dá se dostát k prvku pomoci indexu

# ADT - co to je, které znáte, výhody, zákl. operace nad ADT kontejnery

Abstraktní datové struktury nebo Dynamické datové struktury

rozsah může měnit

last resource

**Definice:** abstraktní datový typ je implementačně nezávislá specifikace struktury dat s operacemi povolenými na této struktuře.

implementačně nezávislá nezávisí na tom jak je to nakódované

Hlavní výhoda ADT:

snadné používání pomocí metod či funkcí

vlastní implementace je programátorovi skryta

jsou implementovány v knihovnách nebo přímo v jazyce

lze označit za kontejnery, určený k organizovanému skladování jiných objektů, podle určitých pravidel.

kontejner by měl mít definované tyto operace: new konstruktor, size, read, insert, delete, clear

# Kritéria pro rozčlenění ADT, typ ukazatel

Počet a vzájemné uspořádání složek

statické= nemění se

dynamické= mění se

Typ objektůuvnitř

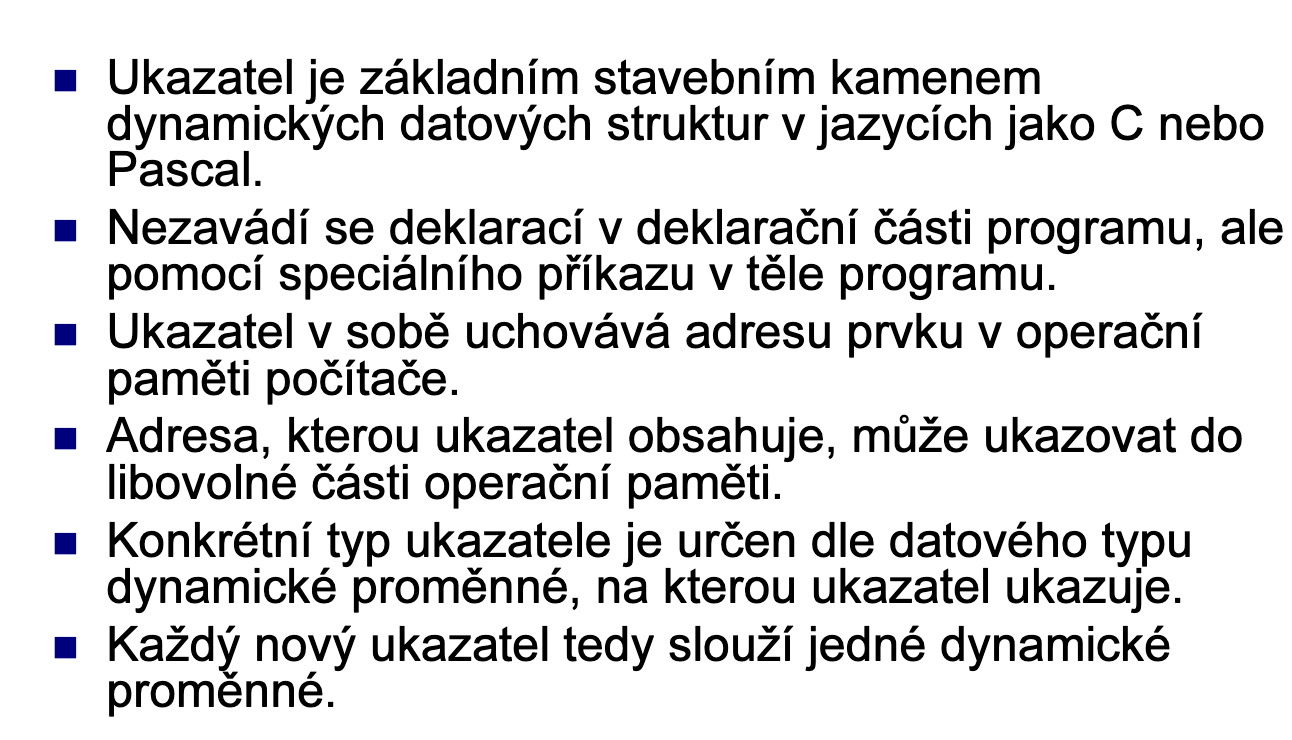
homogenní = všechny stejného typu

nehomogenní= různého typu java ArrayList?

Existuje bezprostřední následník

lineární= existuje [pole, seznam]

nelineární= neexistuje [strom, tabulka]

Pointer = ukazatel

# Popište LIFO, FIFO - vše o co nich víte

Zásobník -LIFO

homogenní, lineární, dynamický

Přístup pouze k vrcholu, vkládání pouze na vrchol.

LIFO= Last-In-First-Out

Využití–odložení informace, výběr v opačném pořadí.

Zásobník lze implementovat pomocí pole či pomocí lineárního seznamu (linked-list).

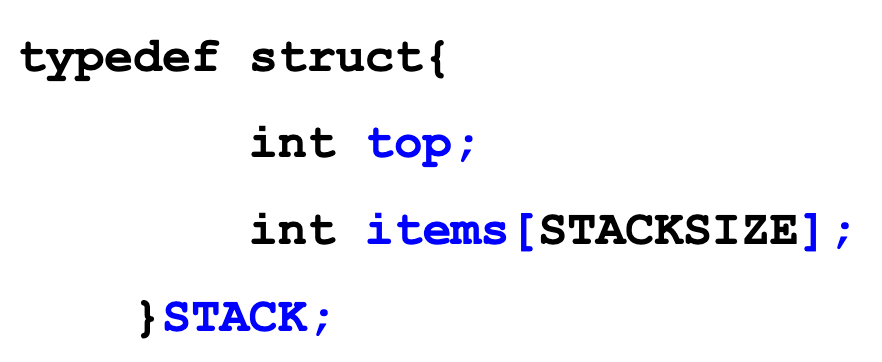
LIFO Operace

create

push

pop

top,is\_empty

Implementace v C

Fronta FIFO

Fronta je opakem zásobníku

homogenní, lineární, dynamická

Prvky se z fronty odebírají v tom pořadí, v jakém se do fronty vkládají

FIFO - First-In-First-Out (kdo dřív přijde..)

Frontu lze implementovat stejně jako zásobník.

Operace

create

push

pop

is\_empty

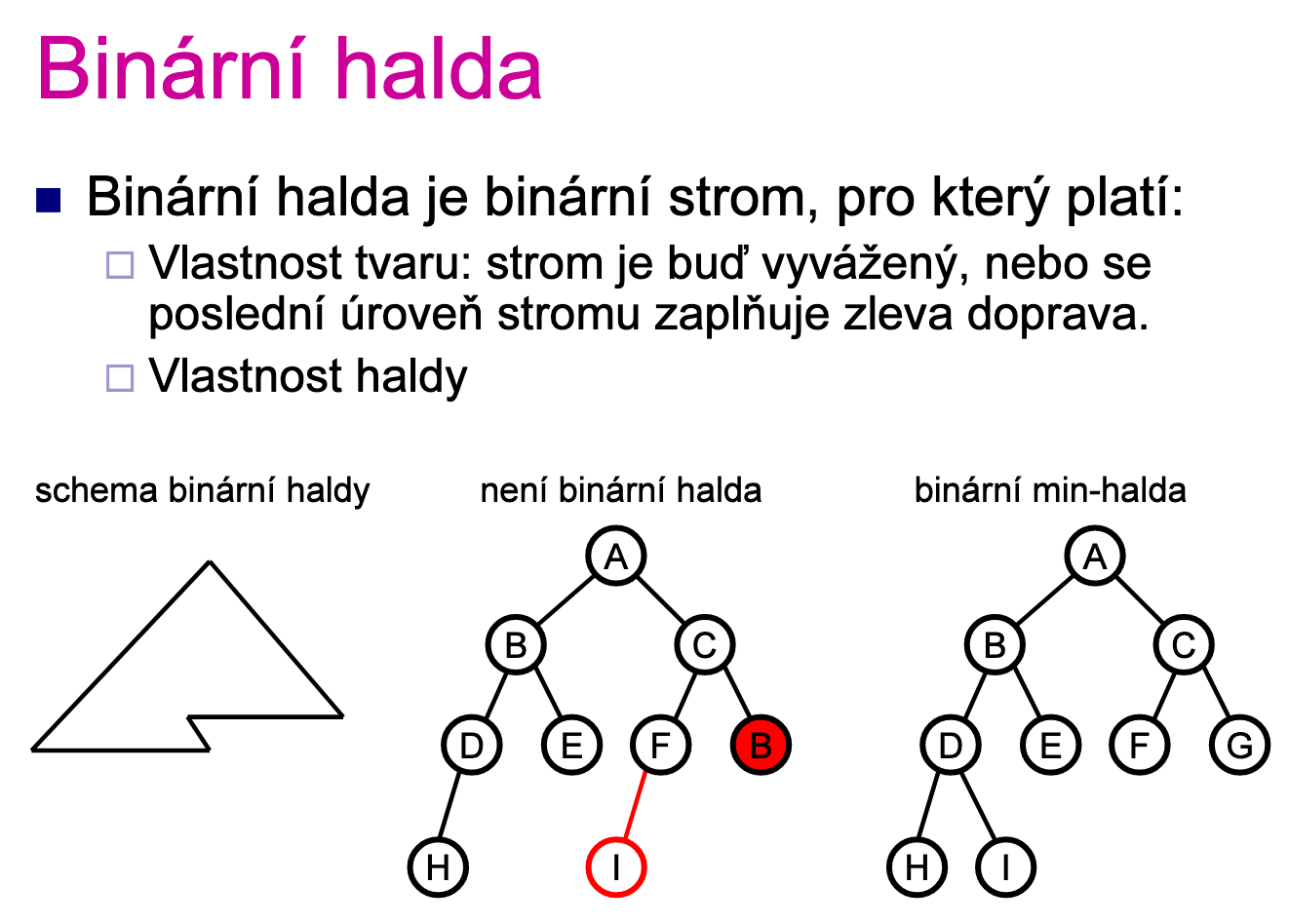
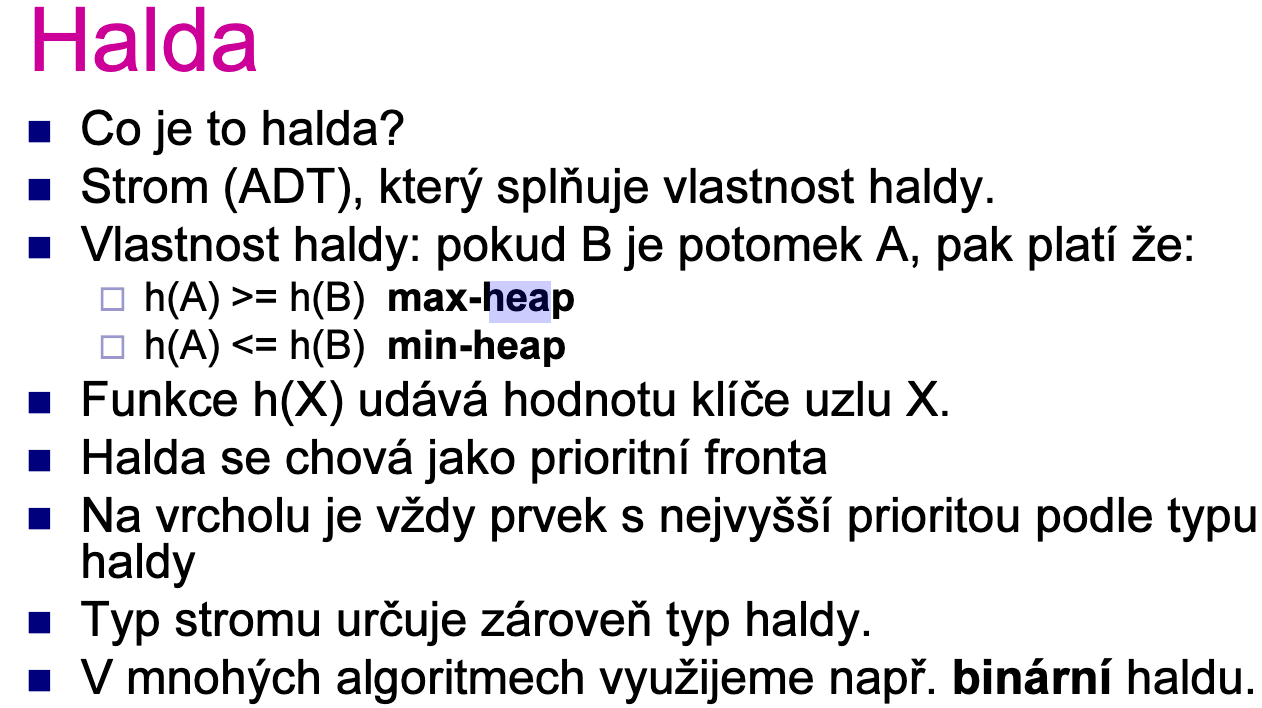
# Popište Kruhový buffer - princip, vlastnosti,možnost použití atd.

Miň práce s paměti

# Popište ADT Seznam, rozdíl oproti LIFO (resp. FIFO)

Operace create, add, remove, is\_empty list, find(element), size, clear, move element

# Popište ADT Heap - princip, vlastnosti, možnost použití

Heap = Halda, Hromada

# Popište ADT Dynamické pole, srovnání, princip, výhody atd.

# ADT Množina - co o něm víte

# Definice algoritmu, jeho vlastnosti

Definice: algoritmus je postup, jak řešit daný problém, abychom realizací tohoto postupu dospěli od zadaných vstupních dat k po žadovanému výsledku

Postupujeme po krocích, které se nazývají příkazy.

Hromadnost (obecnost)

pracuje nad obecnou množinou dat

Determinovanost(určitost)

každý stav algoritmu je jednoznačně určen z výsledků předchozího stavu.

Konečnost

pro konečnou množinu dat, dojde v rozumném čase k výsledku.

Rezultativnost(korektnost)

# Co víte o složitosti algoritmu ?

Ukazuje jak je připraven na práci s různým množství dat.

Jak po čítat slo žitost?

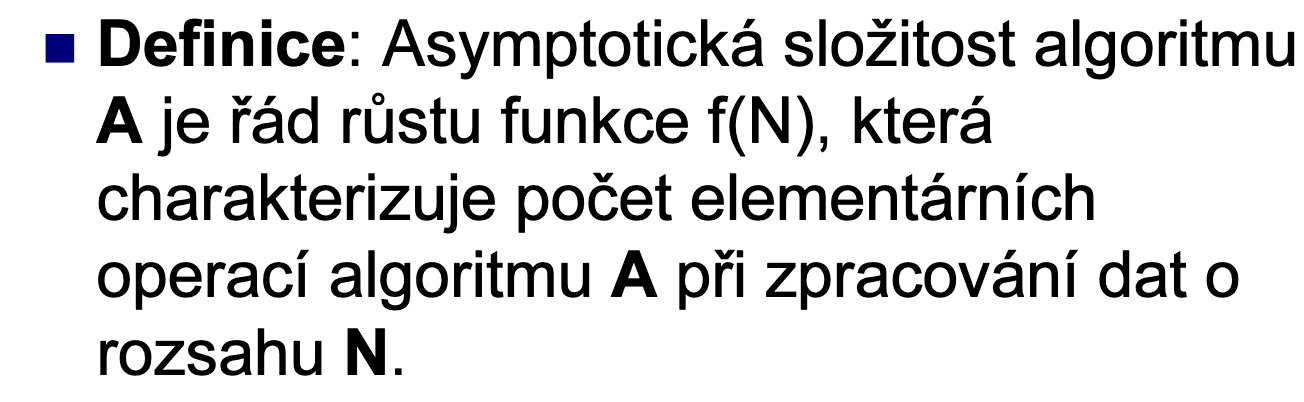
Se číst všechny elementární operace

porovnání dvou čísel

aritmetická operace

přesun čísla v paměti

Zjednodušení 1: počítáme pouze elementární operace nad daty

Zjednodušení 2: počítáme pouze porovnání čísel nad daty

# Co víte o rekurzi a backtracingu?

**Rekurze - definice**

Matematická:

pojmem rekurze rozumíme definování funkce nebo procesu pomocí jeho samého.

Programátorská:

rekurzivní algoritmus obsahuje opakované vnořené volání stejné funkce. Tuto funkci nazýváme rekurzivní.

Tři pravidla:

1.Musí být definována podmínka pro ukončení algoritmu.

2.V každém kroku rekurze musí dojít ke zjednodušení problému.

3.V algoritmu se nejprve musí ověřit, zda nenastala koncová situace. Když ne, provede se rekurzivní krok.

Rozlišujeme dva typy rekurze

Přímá rekurze: funkce volá sama sebe.

Nepřímá rekurze: vzájemné volání několika funkcí. př.: funkce A>B>C>A.

**Backtracking(česky: prohledávání s návratem)**

Slovní formulace: zkoušíme přípustná umístění dam, pokud nejde další dáma umístit, vrátíme se do předchozího stavu a zkusíme další možnost.

# Sorting - co to je, jejich dělení podle typu dat, BogoSort, zajíci a želvy

sort = tříděni, uspořádaní dat.

**Tři „typy “algoritmů**

Školní–jednoduché algoritmy, obvykle kvadratická složitost. Vhodnépro malé množiny dat a pro pochopení principu řazenídat.

Praktické–algoritmy používanév praxi, složitost lepší nežkvadratická, často podpora ve standardní knihovně jazyka.

Teoretické–vědecké práce, které(zatím) do praxe nepronikly.

Typ (velikost) dat

Vnitřní používáme pro data, která lze najednou uchovat v operační paměti.

Vnější v případě rozsáhlých dat na čítaných průběžně z disku.

Strukturovaná data

Stabilní algoritmy –vzájemné pořadí údajů se stejným klíčem zůstane zachováno.

Nestabilní alg. –toto pořadí nelze zaručit.

Částečně seřazená data

V řadě případů jsou data částečně uspořádaná předem.

Přirozený algoritmus –je na takových datech rychlejší.

Nepřirozený algoritmus –je na nich stejně rychlý jako na náhodných datech.

Zajíci a želvy různá čísla se posouvají na svá místa s různou rychlostí.

# Popište princip ShakerSortu a HeapSortu

HeapSort = Binární halda

# Popište princip SelectSortu a BubbleSortu

**SelectSort**

Velmi jednoduchý algoritmus

Snadná implementace

Složitost algoritmu je vždy kvadratická O(N^2)

Vnitřní, nestabilní, nepřirozená

pracuje na principu nalezení minimálního prvku v nesetříděné části posloupnosti a jeho zařazení na konec již setříděné posloupnosti.

# Popište princip InsertSortu a QuickSortu

InsertSort

přirozený algoritmus třídění karet na ruce jednoduchá implementace

efektivní na malých množinách

Složitost –nejhoršíO(N^2), nejlepší O(N)

dokáže řadit data tak, jak přicházejí na vstupu –online algoritmus

Vnitřníi vnější, přirozený, stabilní

efektivní způsob testování, zda jsou data uspořádaná nebo ne

Princip

Pracuje na principu vkládání prvku na jeho správné místo v posloupnosti.

K tomu využívá pomocný prvek, zpravidla nultý prvek posloupnosti.

QuickSort

Nejhoršípřípad O(N2), nejlepší a průměrný O(N log N).

Logaritmickou složitost nelze zaručit, ale reálné aplikace a testy ukazují, že na (pseudo)náhodných datech je vůbec nejrychlejší ze všech obecných řadicích algoritmů.

Díky D&C je dobře paralelizovatelný.

Vnitřní, nestabilní, nepřirozený

Princip

Pracuje na principu rozdělení pole řazených prvků na dvě části(miň než pivot a vic) a tyto potom seřadit.

Využívá rekurzi

# Popište princip MergeSortu a CombSortu

MergeSort

Logaritmická složitost O(N log N) -vždy

Většípaměťo vénároky –obvykle potřebuje odkládací ADT o velikosti N.

Výhoda –stabilní, paralelizovatelný, vyšší výkon na sekvenčních médiích

Implicitní řazenív řadě jazyků–např. GNU C a Java.

Rozdělit na jednotlivé prvky a pak uspořádané sloučit do posloupností a pak sloučit ty posloupnosti.

CombSort

BubbleSort který porovnává prvky vzdálené o mezeru která začíná na N\*3/4 a pak se po každé iteraci zmenšuje o 25%, až na BS.

# Vyhledávání, oč jde, zákl. pojmy, rozdíl mezi lieárním a binárním vyhledáváním

Vyhledávání = Search hledáme klíč k v množině S

Typ prohledávaného prostoru určuje jaký algoritmus zvolíme i jak ho musíme implementovat

Statický

velikost v čase je konstantní

snadno se implementuje (sestavuje)

změna (přidání, odebrání prvku) vytvoří novou verzi prostoru

příklady: telefonní seznam, slovník (kniha), některé ADT např. tuple v Pythonu

Dynamický

velikost se v čase mění

implementace operací je náročnější

příklady: ADT jako slovník, seznam a další

Lineární vyhledávání

jednoduchý a často používaný způsob

postupně procházíme všechny prvky S dokud nenajdeme k

lineární složitost O(n)

pracuje na obecné (neseřazené) množině

Binární vyhledávání

metoda půlení intervalu

S musí být uspořádaný

rekurzivní algoritmus D&C

logaritmická složitost O(log n)

Pokud hledáme jednou v neseřazeních datech tak lineární, jinak musíme promyslet není-li lepší uspořádat množinu a použit binární vyhledávání.

# Binární a interpolační vyhledávání

Interpolační vyhledávání

Varianta binárního

Snaha simulovat lidské chování, např. vyhledávání ve slovníku

Na rozdíl od binárního v. nepočítáme medián, ale odhad dle vzorce:

Index = fist\_index + ((first\_index-last\_index)/(S[first\_index]- S[last\_index])) \* (k - S[first\_index])

# Binární vyhledávací stromy - vlastnosti, metody procházení, vkládání a odstranění prvku

vlastnosti:

BVS je binární strom

hodnota klíče uzlu x = v(x)

pro všechny uzly levého podstromu L platí, že v(L) < v(x)

pro všechny uzly pravého podstromu R platí, že v(R)> v(x)

BVS –využití a vlastnosti

hledání určitého klíče

hledání minima a maxima

řazení dat

inorder BVS zobrazí seřazenou množinu

pomalejší než heapSort (QS, MS)

efektivní metoda online řazení

Odstranění uzlu

Pokud má 2 potomky tak ho vyměníme za největší v levém podstromu nebo z nejmenší v pravém podstromu.

Když má jednoho potomka tak ho tím potomkem nahradíme

Když nemá žádného potomka tak stačí proste odstranit.

# Vyvažování stromů - co to je, proč to je, co je bal a rotace

Proč? Vyvážený strom má vyhledávání se složitostí log(n).

Rozdílem je podle čeho určujeme, že je strom vyvážený:

**výška** podstromů -AVL strom (samovyvažující binární strom, hloubka levého a pravého podstromu se liší nejvýše o 1)

**výška**+ počet potomků –varianty B-stromu, př. 2-3 strom

**váha** podstromů (počty uzlů) -váhově vyvážený strom

**Bal je faktor vyváženosti** bal(u) = h\_left - h\_right

Rotace je operace která slouží k vyvážení stromu. L ,R, LR, RL

# Hashování - základní terminologie, princip, asociativní a adresní vyhledávání, hašovací funkce + kolize

Hash = otisk, miniatura

Asociativní vyhledávání

Hledáme klič ve stromu

Složitost je O(log N)

Adresní vyhledávání

Přímé:

Hledaný klíčje přímo indexem, adresou v paměti.

Počet klíčůurčuje velikost indexu – náročné na paměť

Složitost elementární O(1)

Hašováním:

Adresu v paměti vypočteme z hledaného klíče

Průměrná složitost je opět O(1)

Hašovací funkceh(k)

Definice: hašovací funkce h(k) je zobrazením z množiny klíčů Kdo množiny adres A= <Amin, Amax>.

Hašovací funkce h(k)

Je silnězávislá na vlastnostech klíčů a jejich reprezentaci v paměti

Ideální funkce:

výpočetně co nejjednodušší (rychlá)

aproximuje náhodnou funkci

využívá rovnoměrně adresní prostor

generuje minimum kolizí

Kolizí nazýváme stav kdy pro dva různé klíče k1≠ k2 platí, že h(k1) = h(k2)

# Zřetězené hashování, otevřené hashování, linear probing, double hashing - principy, výhody/nevýhody atd.

Zřetězené hašování

Adresy v hašovací tabulce obsahují lineární seznamy

V případě kolize (stejná adresa) se prvek vloží na konec seznamu

V případě hledání sekvenčně procházíme konkrétní seznam

Otevřené hašování

Open adress hashing

Tabulka adres uložená do pole

V případě kolize prohledáváme určitou metodou další prvky pole, dokud nenajdeme prázdnou pozici.

Při vyhledávání postupujeme stejně–stejnou metodou procházíme, pokud najdeme volnou pozici znamená to že prvek není indexován.

Podle metody hledání volného místa rozlišujeme:

1. lineární prohledávání (linear probing)

2. dvojí hašování (double hashing)

Linear probing

h(k) = [(k mod 5) + i ] mod 5 = (k + i) mod 5 kde 5 je velikost pole, i = 0 na začátku

pokud dojde ke kolizí i++

Dvojité hešování

Double hashing

Na rozdíl od lineárního prohledávání zde jako metodu použijeme druhou hašovací funkci

Obě jsou funkcí k

Každá má jinou sekvenci prohledávání

h(k) = [h1(k) + i.h2(k) ] mod m

# Prohledávání řetězců - terminologie, princip, přirozené prohledávání, KMP, chybová funkce

řetězec T (text) a vyhledávaný řetězec P (vzor, pattern)

Podřetězec S[i:j] ječást řetězce S mezi indexy i a j.

Prefix(předpona) S je podřetězec S[0:i]

Suffix(přípona) S je podřetězec S[i:m-1]. Kde i je libovolný index mezi 0 a m-1

Každé slovo je prefixem i suffixem sebe sama – takový prefix nazýváme nevlastní.

Abeceda A je konečná množina znaků ze kterých tvoříme T a P.

Velikost abecedy je během algoritmu konstantní.

Přirozené prohledávání

Postupně procházíme celý řetězec T a pro každou pozici testujeme, zda na ní nezačíná hledaný řetězec P.

Složitost algoritmu je O(mn) – nejhorší případ.

V praxi při hledání v běžném textu obvykle dosahujeme O(m+n).

Algoritmus je rychlý, pokud je abeceda textu „velká“např. A..Z, a..z, 1..9, atd.

Algoritmus je pomalý pro „malou“ abecedu

Tedy hlavně 0,1 (binární soubory, obrázkové soubory, atd.)

Knuth-Morris-Pratt (KMP)

Princip je stejný jako u přirozeného prohledávání

Řízení procesu:

Nastupuje pokud se vyskytne neshoda mezi textem a vzorem v pozici P[j]

Jaký je největší možný posun vzoru abychom se vyhnuli opakovanému porovnávání?

Složitost O(m+n)

Algoritmus se nikdy nevrací ve vstupním textu – vhodné pro sekvenční zpracování rozsáhlých dat.

Chybová funkce

F(k) velikost největšího prefixu, který je zároveň sufixem, k je pozice kde došlo k neshodě v Patternu

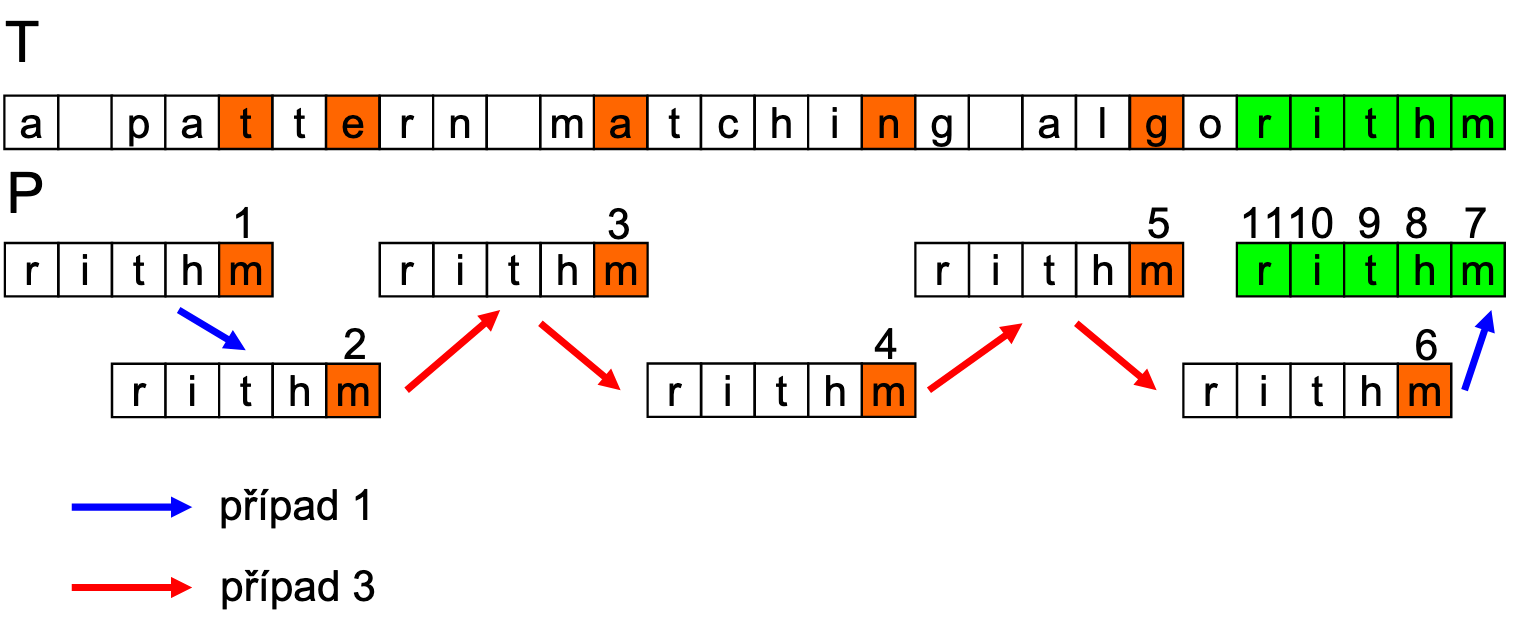
# Prohledávání řetězců - Boyer-Moore, Rabin-Karp - princip, srovnání

Boyer-Moore

Časová složitost BM algoritmu je v nejhorším případě O(nm + A)

Algoritmus je rychlejší pokud je abeceda (A) velká, pomalý pro malou abecedu.

Tedy stejně jako přirozené prohledávání (PP) je vhodný pro text, špatný pro binární vstupy

BM je rychlejší než PP v případě vyhledávání nad stejnou abecedou.

Rabin-Karp algortimus

Založen na použití hašování

Vypočteme hash pro vzor P (délky m) a pro prokaždý podřetězec řetězce T délky m.

Procházíme řetězcem T ale místo jednotlivých znakůporovnáváme hash každého podřetězce a vzoru.

V případě shody provedeme test podřetězce a vzoru znak po znaku – ochrana proti kolizi haše.