1 - 4

Jak funguje metoda Rozděl a panuj?

:r1 Dělí problém na dílčí úlohy, které musí být nezávislé.

:r2 Dělí problém na dílčí úlohy, které mohou být závislé.

:r3 Rozdělí úlohy mezi více počítačů

:r4 Nehledá konkrétní řešení, ale jen nějaké vhodné přiblížení

:r1 ok 2

--

Jak funguje hladový algoritmus?

:r1 Volá sám sebe, dokud nenalezne řešení

:r2 Provádí rozhodnutí založená na náhodných (pseudonáhodných) volbách

:r3 Hledá lokální extrémy ve snaze najít extrém globální

:r4 Dělí problém na dílčí úlohy, které jsou závislé.

:r3 ok 2

--

Co je to ADT – Abstraktní datový typ?

:r1 Abstraktní data, která nelze implementovat

:r2 Označení pro typy dat, která jsou nezávislá na vlastní implementaci

:r3 Obecné datové typy jako Interger, Real, Boolean…

:r4 Konkrétní implementace datového typu

:r2 ok 2

--

Základní operace s ADT jsou:

:r1 Konstruktor, Selektor, Modifikátor

:r2 Konstruktor, Destruktor

:r3 Konstruktor, Selektor, Modifikátor, Destruktor

:r4 Selektor, Destruktor

:r1 ok 2

--

Experimentální analýza časové náročnosti algoritmu:

:r1 Probíhá v prostředí, kde běží daný program (algoritmus)

:r2 Nezávisí na prostředí, kde program běží

:r3 Nevyžaduje implementaci algoritmu

:r4 Zaměřuje se na nejjednodušší (nejméně náročný) případ

:r1 ok 2

--

Pseudo-code:

:r1 Závisí na programovacím jazyku

:r2 Odhaluje problémy konkrétní implementace

:r3 Je vyšší úroveň popisu, více strukturovaný než klasický popis, méně detailní než implementace

:r4 Nižší úroveň popisu, méně strukturovaný než klasický popis, vyžaduje znalosti konkrétního programovacího jazyka

:r3 ok 2

--

Primitivní operace

:r1 Závisí na konkrétním programovacím jazyku, nelze ji identifikovat v pseudokódu

:r2 Jiný název pro konkrétní implementaci procedury (metody, algoritmu)

:r3 Blok programu zahrnující několik operací, které jsou prováděny na základě jednoho volání

:r4 Základní operace provedená algoritmem jako je vyhodnocení výrazu, přiřazení hodnoty do proměnné, volání procedury…

:r4 ok 2

--

Jestliže je počet provedených algoritmem 2n²-3, pak časová náročnost a Big O notace je:

:r1 Exponenciální, O(c^n -3)

:r2 Logaritmická, O(log n)

:r3 Konstantní, O(1)

:r4 Kvadratická, O(n^2)

:r4 ok 2

--

ADT Zásobník

:r1 Vkládání a mazání probíhá pomocí principu FIFO

:r2 Základní operace jsou: push(Object), findMin(), findMax(), deleteAll()

:r3 Nikdy nevyhodí žádnou výjimku

:r4 Řídí se principem LIFO a základní operace jsou: push(Object), pop()

:r4 ok 2

--

Pokud je Zásobník založen na poli,

:r1 Pak časová náročnost každé operace je O(n^2 +n), je-li n počet prvků v zásobníku

:r2 Na začátku musíme definovat velikost zásobníku

:r3 Lze uchovávat libovolný počet prvků, vždy lze jednoduše zásobník zvětšit

:r4 Při přidání prvku do plného zásobníku vyvolá výjimku EmptyStackException

:r2 ok 2

--

ADT Fronta

:r1 Vkládání a mazání probíhá pomocí principu FIFO

:r2 Základní operace jsou: push(Object), findMin(), findMax(), deleteAll()

:r3 Vkládání a mazání probíhá pomocí principu LIFO

:r4 Nikdy nevyhodí žádnou výjimku

:r1 ok 2

--

Fronta

:r1 Uchovává kolekci položek, kde položka je uspořádanou dvojicí (priorita-hodnota), na jejichž základě udržuje pořadí ve frontě

:r2 Pro implementaci využíváme kruhové pole

:r3 Základní operace jsou: enqueue(object), dequeue(), first(), last(), order()

:r4 Časová složitost pro libovolnou implementaci je nejlépe O(n^2 log n)

:r2 ok 2

--

Vektor

:r1 Definuje vztah před/po mezi pozicemi

:r2 Prvky jsou uspořádané podle velikost

:r3 Rozšiřuje pojem pole o ukládání sekvence libovolných objektů

:r4 Umožňuje ukládat objekty i na záporné pozice (záporný index)

:r3 ok 2

--

Seznam

:r1 Dynamická datová struktura, velikost se mění na základě přidání (odebrání) prvku

:r2 Statická datová struktura, velikost se nemění

:r3 Prvek může být čten, vkládán a odebírán pomocí určení jeho pořadí

:r4 Základní, obecně použitelný datový typ pro ukládání uspořádaného souboru prvků

:r1 ok 2

--

Jednosměrný spojový seznam (Single linked list)

:r1 Uzel obsahuje odkaz na předchozí i následující uzel

:r2 Lze ho použít pro implementaci zásobníku a fronty s lineární paměťovou náročností a konstantní časovou náročností pro každou základní operaci

:r3 Statická datová struktura

:r4 Definuje vztah před/po mezi pozicemi

:r2 ok 2

--

Sekvence

:r1 Je spojení Vektoru a Seznamu, přístup k prvkům pomocí pořadí i pozice

:r2 K prvkům nelze přistupovat pomocí určení jejich pořadí

:r3 Přístup k prvkům na základě principu FIFO

:r4 Přístup k prvkům na základě principu LIFO

:r1 ok 2

5 – 8

Strom

:r1 Lineární statická datová struktura

:r2 Obsahuje uzly, mezi nimiž je vztah předek-potomek (rodič-dítě)

:r3 Externí uzel se nazývá kořen

:r4 Interní uzel se nazývá list

:r2 ok 2

--

Binární strom

:r1 Obsahuje dva kořeny

:r2 Obsahuje právě dva listy

:r3 Každý interní uzel má nejméně dva potomky, z nichž alespoň jeden je tvořen listem

:r4 Každý interní uzel má právě dva potomky, jež tvoří uspořádanou dvojici (levý potomek, pravý potomek)

:r4 ok 2

--

Prioritní fronta

:r1 Zavádí pomocný datový typ Comparator, který umožňuje porovnávat dva objekty

:r2 Dva rozdílné prvky musí vždy mít rozdílnou prioritu (klíč)

:r3 Na množině prvků musí být definováno uspořádání

:r4 Základní operace s prioritní frontou jsou: insertItem(k,o), removeMin(), upheap(), downheap()

:r1 ok 2

--

Prioritní fronta

:r1 K uloženým prvkům lze přistoupit na základě jejich indexu ve frontě

:r2 Klíče jsou libovolné objekty, na nichž lze definovat pořadí, dva rozdílné prvky mohou mít stejný klíč

:r3 Lineární statická datová struktura

:r4 Prvek s nejmenší prioritou je vždy odebírán jako první

:r2 ok 2

--

Halda

:r1 Je reprezentována jako binární strom, který uchovává klíče jako interní uzly, pro něž platí, že klíč uzlu je vždy větší nebo roven klíče rodiče

:r2 Kořen obsahuje největší klíč

:r3 Poslední uzel hromady je externí uzel nacházející se nejvíce vlevo

:r4 Funkce downheap() obnoví uspořádání haldy při vkládání nového prvku

:r1 ok 2

--

Halda

:r1 Přístup k prvkům na základě principu FIFO

:r2 Přístup k prvkům na základě principu LIFO

:r3 Po vložení uzlu je potřeba volat funkci upheap(), která obnoví uspořádání hromady

:r4 Vkládání a odebírání libovolného prvku z haldy nenaruší její uspořádání

:r3 ok 2

--

Slovník

:r1 Velikost je nutné nastavit vždy při vytváření slovníku, nezávisle na implementaci

:r2 Vyžaduje, aby ve dvojici klíč-hodnota, bylo definováno uspořádání jak na klíči, tak i na hodnotě

:r3 Uspořádaná dvojice klíč-hodnota, kde klíče jsou unikátní, hodnoty nikoli

:r4 Vždy odebíráme prvek s nejmenším klíčem

:r3 ok 2

--

Log File

:r1 Nelze použít, pokud na množině hodnot není definováno uspořádání

:r2 Používá se pro malé slovníky, nebo aplikace, kde je nejčastější operací vyhledávání

:r3 Využívá se tam, kde je nejčastější operací vkládání, kdežto vyhledávání a odebírání se provádí zřídka.

:r4 Prvky vkládáme vždy doprostřed sekvence

:r3 ok 2

--

Vyhledávací tabulka

:r1 Pro implementaci vždy využíváme neuspořádaný slovník

:r2 Používá se pro malé slovníky, nebo aplikace, kde je nejčastější operací vyhledávání

:r3 Využívá se tam, kde je nejčastější operací vkládání, kdežto vyhledávání a odebírání se provádí zřídka

:r4 Přístup k prvkům na základě principu LIFO

:r2 ok 2

--

Hashovací tabulka

:r1 Pro daný typ klíče obsahuje hashovací funkci, která klíči přiřazuje celočíselnou hodnotu, a pole (tabulku) o velikosti N

:r2 Nelze použít pokud hash funkce vrátí na různé klíče stejnou hodnotu (dojde ke kolizi)

:r3 Vyžaduje, aby ve dvojici klíč-hodnota, bylo definováno uspořádání jak na klíči, tak i na hodnotě

:r4 Uspořádává hodnoty podle klíče, nezávisle na hodnotě hashovací funkce, ta slouží pouze pro kontrolu uložených dat

:r1 ok 2

--

Řadící algoritmy

:r1 Označujeme jako stabilní, pokud zachovávají pořadí vkládání položek se stejným klíčem

:r2 Řadí soubor dat podle hodnoty, nikoli klíče

:r3 Vyžadují definované uspořádání na hodnotách

:r4 Nelze použít, pokud data obsahují několik klíčů se stejnou hodnotou

:r1 ok 2

--

Bubblesort

:r1 Pracuje tak, že porovnává první a poslední prvek řady

:r2 Vyžaduje dodatečnou pomocnou paměť o velikosti řazeného datového souboru

:r3 Pro porovnání libovolného souboru dat stačí jediný průchod

:r4 Univerzální, prvky s největší hodnotou probublávají na konec seznamu

:r4 ok 2

--

Heap sort

:r1 Je stabilní řadící algoritmus využívající ADT Seznam

:r2 Z pole vytvoří haldu, kdy nejmenší prvek umístí jako kořen, který následně odebereme

:r3 Náročný, jeho minimální časová složitost je O(n^3)

:r4 Nedokáže řadit data na místě, vyžaduje dodatečnou pomocnou paměť o velikosti řazeného datového souboru

:r2 ok 2

--

Insertion sort

:r1 Porovnává vždy dva sousední prvky v posloupnosti

:r2 Selhává, pokud je množina částečně seřazená

:r3 Stabilní, efektivní na částečně seřazených množinách, dokáže řadit data online (tak, jak přicházejí na vstup)

:r4 Nestabilní, využívá se pro velké množství dat

:r3 ok

9 – 12

Merge sort

:r1 Nestabilní, nelze ho paralelizovat, potřebná paměť navíc je O(1)

:r2 Stabilní, paralelizovatelný, nejhorší časová složitost je O(nlog n)

:r3 Pro porovnání libovolného souboru dat stačí jediný průchod

:r4 Pracuje tak, že porovnává první a poslední prvek řady

:r2 ok 2

--

Quicksort

:r1 Stabilní, přímý, pro porovnání libovolného souboru dat stačí jediný průchod

:r2 Selhává, pokud je množina částečně seřazená

:r3 Univerzální, prvky s největší hodnotou probublávají na konec seznamu

:r4 V průměru jde o nejrychlejší známý algoritmus, nestabilní, pro třídění využívá pivot

:r4 ok 2

--

Selection sort

:r1 Algoritmus složitý na implementace, ale velmi rychlý, pro porovnání libovolného souboru dat stačí jediný průchod

:r2 Univerzální, lokální, nestabilní, vhodný pro malé množství dat

:r3 Vyžaduje další paměť o velikosti O(n^2)

:r4 Minimální a průměrná časová složitost je O(n)

:r2 ok 2

--

Pro řadící algoritmy platí:

:r1 V nejhorším případě je časová složitost Heapsortu a Mergesortu O(nlog n)

:r2 Heapsort, Quicksort a Insertion sort jsou všechny nestabilní

:r3 Heapsort, Quicksort, Mergesort a Insertion sort řadí data na místě (in-place, dodatečná paměťová náročnost je O(1))

:r4 Bubble sort a Insertion sort mají v ideálním případě časovou složitost konstantní - O(1)

:r1 ok 2

--

Pattern matching

:r1 Znamená hledání vzoru P v dané sekvenci T (hledání podřetězce v řetězci)

:r2 Porovnání dvou objektů na základě jejich velikosti

:r3 Nalezení opakujících se částí textu

:r4 Nalezení sekvence, kterou lze daný text zakódovat

:r1 ok 2

--

Pattern matching – Brute force

:r1 Prochází text zprava doleva (odzadu), neprochází všechny pozice

:r2 Porovnává vzor P s textem T pro všechny možné pozice

:r3 Je jednoduchý a časově nenáročný algoritmus, jeho náročnost s delším textem klesá

:r4 Odebírání znaků z textu, dokud z textu T nezůstane řetězec stejné délky jako P

:r2 ok 2

--

Pattern matching – KMP algoritmus

:r1 Prohledává text zleva doprava, podobně jako Brute-Force algoritmus dělá všechna porovnání

:r2 Pokud narazí na neshodu, posune se hledání o více než jedno písmeno, nedělá všechna možná porovnání

:r3 Je jednoduchý a časově nenáročný algoritmus, jeho náročnost s delším textem klesá

:r4 Porovnává řetězce odzadu, posouvá se vždy o jeden znak

:r2 ok 2

--

Trie

:r1 Pořadí uzlu na dané úrovni označuje pořadí písmene ve slovu

:r2 Pokud je Trie komprimovaný, každý uzel obsahuje právě jedno písmeno

:r3 Struktura sloužící pro zpracování textu, kde v každém uzlu je jedno písmeno, či slovo

:r4 Nelze do něj uložit celý text

:r3 ok 2

--

Teorie grafů

:r1 Pokud je graf orientovaný, obsahuje alespoň jednu orientovanou hranu (ne všechny hrany musí být orientované)

:r2 Sled je posloupnost vrcholů taková, že mezi po sobě jdoucími vrcholy existuje hrana a vrcholy se neopakují

:r3 Hrana je vždy určena více než dvěma vrcholy, váhou a směrem

:r4 Graf je uspořádaná dvojice V, E, kde V je množina vrcholů a E je množina hran

:r4 ok 2

--

Hledání nejkratší cesty

:r1 V grafu probíhá v nejhorším případě vždy s časovou složitostí O(n) nezávisle na vybraném algoritmu

:r2 Pomocí Dijkstrova algoritmu funguje na libovolném typu grafu s libovolnými hodnotami hran

:r3 Pomocí Floydova-Warshallova algoritmu vyžaduje orientovaný graf s kladnými hranami a nalezne nejkratší cestu mezi všemi vrcholy

:r4 Nelze použít na grafu, který obsahuje záporné hrany

:r3 ok 2

--

Genetické algoritmy

:r1 Napodobují techniky evoluční biologie, patří do umělé inteligence

:r2 Je přesný deterministický algoritmus, nevyužívá žádné náhody

:r3 Způsob popsání (zakódování) individuí nemá vliv na úspěch, či neúspěch řešení konkrétní úlohy

:r4 Pokud jsou spuštěny několikrát po sobě, vždy nutně dávají stejný výsledek

:r1 ok 2

--

Selekce – výběr jedinců pro rodičovství

:r1 Je prováděna tak, že rodiči se stanou jedinci s nejmenší fitness hodnotou

:r2 Při náhodném výběru rodičů využíváme znalosti jejich fitness hodnoty

:r3 Je prováděna, abychom získali nevýhodnější jedince pro další generaci

:r4 Nemá vliv na úspěch, či neúspěch řešení konkrétní úlohy

:r3 ok 2

--

Křížení

:r1 Neexistuje možnost, jak v jedné generaci zkřížit více než dva jedince

:r2 Rodiče si vymění část genetického kódu, při jednobodovém křížení vzniknou dva noví jedinci

:r3 Každý jedinec v dané generaci se musí stát rodičem

:r4 Rodič vždy zaniká a jeho místo zaujme potomek, tento potomek bude mít stejnou fitness hodnotu jako jeho rodič

:r2 ok 2

--

Mutace

:r1 Náhodná změna v genomu jedince, probíhá s velmi malou pravděpodobností

:r2 Nikdy nepřináší nové vlastnosti, které by se nevyskytovali v původní generaci

:r3 Každý jedinec z každé generace zmutuje alespoň na jednom místě

:r4 Mutace u různých jedinců probíhají vždy na jednom a tom samém místě v genomu

:r1 ok 2