Vlastnosti datových struktur - Seřazenost a opakování prvků, Indexace, hashování a klíče prvků

Datové struktury
Datové struktury jsou specializované formáty pro organizaci, ukládání a manipulaci s daty v paměti počítače. Jejich efektivita a vhodnost závisí na konkrétním použití a požadavcích na operace - hledání, vkládání, mazání atd….

Vlastnosti datových struktur
Seřazenost prvků - určuje předvídatelnost pořadí a možnosti vyhledávání
Opakovatelnost hodnot - definuje jedničnost nebo multiplicitu
Přístupový mechanismus - indexy a klíče jako způsoby adresace
Hashování - transformační princip pro efektivní ukládání a vyhledávání

Seřazené
Automaticky udržují své prvky v definovaném pořadí. Toto pořadí může být vzestupné, sestupné nebo seřazené podle svého.

Na rozdíl od některých jiných jazyků, Python **nemá** ve své standardní knihovně přímo vestavěné seřazené kolekce, které by automaticky udržovaly prvky seřazené. Místo toho Python nabízí několik přístupů:

- Manuální seřazení běžných kolekcí pomocí metod .sort() nebo funkce sorted()
- Externí knihovny jako sortedcontainers, které poskytují seřazené kolekce
- Vlastní implementace seřazených struktur

Výhodou seřazených struktur je rychlejší vyhledávání. Jelikož jsou prvky seřazené, můžeme použít efektivní algoritmy jako binární vyhledávání s časovou složitostí O(log n) namísto lineárního vyhledávání O(n). Zároveň máme okamžitý přístup k minimálním a maximálním hodnotám a můžeme efektivně provádět rozsahové dotazy.

Nevýhodou seřazených struktur je pomalejší vkládání a mazání prvků, protože struktura musí vždy zajistit, aby byly prvky správně seřazeny. Tyto operace mají obvykle složitost O(log n) nebo O(n), zatímco u neseřazených struktur mohou být rychlejší.

Typickým příkladem v programovacích jazycích jsou SortedSet, SortedList nebo SortedDictionary

```
# Manualne
list = [1,5,3,8,2,9]
list.sort()
print(list)

# Sortedcontainers
from sortedcontainers import SortedList
list2 = [1,5,3,8,2,9]
sortedlist = SortedList(list2)
print(sortedlist)
```

Neseřazené

Neseřazené datové struktury negarantují žádné specifické pořadí prvků podle jejich hodnoty. Prvky mohou být uspořádány podle pořadí vložení (v některých případech) nebo mohou být v libovolném pořadí určeném vnitřní implementací.

List []

- Neseřazený podle hodnot, ale zachovává pořadí vložení
- Umožňuje duplicitní prvky
- Indexovaný
- Měnitelný

Dictionary {key:value}

- Neseřazený podle hodnot klíčů, od Python 3.7+ zachovává pořadí vložení
- Klíče musí být unikátní, hodnoty mohou být duplicitní
- Přístup podle klíčů
- Měnitelný

Set {}

- Neseřazený, nezachovává žádné pevné pořadí
- Nepovoluje duplicity
- Měnitelný

Tuple ()

- Neseřazený podle hodnot, ale zachovává pořadí vložení
- Umožňuje duplicitní prvky
- Indexovaný
- Neměnný

Indexace
Mechanismus přístupu k prvkům datové struktury pomocí jejich pozice.

Na nejnižší úrovni implementace funguje indexace na principu výpočtu adresy v paměti. Když vytvoříme indexovanou strukturu (jako je list v Pythonu), operační systém alokuje souvislý blok paměti. První prvek má určitou základní adresu v paměti, a každý další prvek je uložen na adrese

Indexaci podporují sekvenční datové typy jako list, tuple a string. Přístup k prvku pomocí indexu má konstantní časovou složitost O(1), což umožňuje efektivní náhodný přístup k datům bez ohledu na velikost struktury. Python rozšiřuje indexaci o slicing (vytváření výřezů) pomocí syntaxe `[start:stop:step]`. Tato technika umožňuje extrahovat podsekvence, například `cisla[1:3]`.

Důležité je vědět, že ne všechny struktury podporují indexaci. Dictionary (slovník) a set (množina) nepoužívají indexaci, ale jiné přístupové mechanismy. Slovník používá klíče místo indexů, a přestože syntaxe `slovnik["klic"]` vypadá podobně jako indexace, jde o zcela jiný princip.

Hlavní nevýhodou indexovaných struktur je neefektivnost vkládání a mazání prvků na konkrétních pozicích. Při vložení prvku doprostřed seznamu musí být všechny následující prvky posunuty, což vede k časové složitosti O(n). Proto pro aplikace s častými vnitřními modifikacemi mohou být vhodnější jiné datové struktury.

Hashování a klíče
Hashování představuje základní princip, který umožňuje velmi rychlý přístup k datům v některých datových strukturách. Jde o proces, při kterém se data libovolné velikosti převádějí na číselné hodnoty fixní délky. Tyto hodnoty, nazývané hashe, slouží jako "otisky" původních dat a zároveň jako adresy pro jejich uložení v paměti.

V Pythonu je hashování implementováno prostřednictvím metody `__hash__`, kterou mají mnohé objekty. Tato metoda vrací celé číslo reprezentující hash objektu. Když použijeme built-in funkci `hash()`, voláme právě tuto metodu. Například `hash("Python")` vrátí celočíselnou hodnotu, která je unikátní pro řetězec "Python".

Hashování je klíčové pro dvě důležité datové struktury v Pythonu: dictionary (slovník) a set (množina). Dictionary ukládá páry klíč-hodnota, zatímco set uchovává pouze unikátní prvky. Obě struktury používají hashovací mechanismus, který jim umožňuje dosáhnout průměrné časové složitosti O(1) pro operace vyhledávání, vkládání a mazání. To je výrazně rychlejší než lineární vyhledávání v seznamech, které má složitost O(n).

V Pythonu jsou hashovatelné všechny vestavěné neměnné typy: čísla, řetězce, tuple (obsahující jen hashovatelné typy). Naopak měnitelné typy jako seznam, slovník a běžná množina hashovatelné nejsou a nemohou sloužit jako klíče.

Aby mohl být objekt použit jako klíč ve slovníku nebo jako prvek v množině, musí být hashovatelný. To znamená, že musí mít implementovanou metodu `__hash__`, musí podporovat porovnání na rovnost (metoda `__eq__`) a měl by být neměnný (immutable). Změna objektu po jeho vytvoření by mohla změnit jeho hash, což by vedlo k nekonzistentnímu chování.