Univerzita Karlova v Praze

Přírodovědecká fakulta

Katedra aplikované geoinformatiky a kartografie

HASHOVÁNÍ: METODA SEPARATE CHAINING V JAZYCE PYTHON

Jáchym Slovák

2. B-FGG

v Chocni 4. 2. 2026

Anotace:

Tato práce se zabývá principem hashovacích tabulek za využití metody Separate Chaining a jejich implementaci v jazyce Python.

Zadání:

Implementujte dynamickou datovou strukturu hashovací tabulka, která ukládá dvojice (klíč, hodnota). Pro řešení kolizí použijte metodu Separate Chaining. Vstupní data budou tvořena textem načteným ze souboru. Hashovací funkce bude mít znak „z“ tvar h(z)=z%10. Pro každý znak uložte informaci, ve kterém koši se bude nacházet. Vypište také, které znaky leží v jednotlivých koších a jaký je jejich počet. Výsledky uložte do textového souboru.

Rozbor problému:

Hashovací tabulka je dynamická datová struktura, která umožňuje rychlé hledání v databázi podle klíčů. Princip hashování stojí na zakódování klíče pomocí hashovací funkce a následném rozřazení do stanoveného počtu košů podle výsledku hashovací funkce. Tímto způsobem může být každá dvojice (případně řádek tabulky) přiřazena ke koši s indexem odpovídajícím výsledku hashovací funkce jejího klíče. Vstupem do samotné funkce musí být číslo. V případě, že klíčem je jiný znak nežli číslo, přistupuje se k převodu znaku pomocí ASCII kódu na číslo. Při hashování mohou vycházet stejné výsledky pro různé znaky – dochází tak ke kolizím. Metoda Separate Chaining tento problém řeší tak, že na jednotlivé koše mohou být navázány separátní řetězce dvojic. Při vyhledávání v databázi s takovouto datovou strukturou nemusí program prohledávat celý seznam dvojic, nýbrž pomocí výpočtu funkce zjistí index koše, ve kterém se hledaný objekt nachází a prohledává pouze obsah jednoho koše.

Existující algoritmy:

Existuje několik variant řešení problematiky kolizí s metodou Separate Chaining. Často využívaným přístupem může být zřetězení pomocí spojového seznamu (Linked List). Každý koš odkazuje na první prvek v seznamu, na který jsou navázány další prvky pomocí uzlů.

Dalším, v jazyce Python často využívaným přístupem je dynamické pole (Dynamic Array). Prvky jsou za sebou řazeny v poli, bez využití uzlů. Tento přístup je v Pythonu velmi efektivní, jelikož využívá jeho dobře vyvinutý seznam.

Pokročilým přístupem je např. zřetězení pomocí samovyvažovacích stromů (Self-balancing Trees). Při velkém počtu kolizí se koše transformují ze seznamů na binární vyhledávací stromy.

Kromě metody Separate Chaining se pro řešení kolizí při hashování dají použít další metody. Například můžeme uvést Linear Probing, který při kolizi hledá nejbližší volné

místo v poli, Quadratic Probing, pomocí kterého se hledá volné místo za použití kvadratické rovnice nebo Double Hashing, kde se využívá druhé hashovací funkce.

Použitý algoritmus:

Zvolený algoritmus implementuje hashovací tabulku s fixním počtem 10 košů, přičemž kolize jsou řešeny metodou Separate Chaining za použití dynamických polí.

1. Třída Hashovací tabulka

Na začátku algoritmu je vytvořena třída *Hashovacitabulka*. V jejím konstruktoru *__init__* dojde k vytvoření 10 prázdných košů (listů) v rámci seznamu *self.kose*. To umožňuje každému koši dynamicky růst s narůstajícím počtem prvků v tabulce. Vzhledem k tomu, že jsou jednotlivé koše součástí seznamu, má každý koš svou danou pozici, tedy index.

Pro určení indexu, do kterého daný prvek patří, je použita hashovací funkce *h(z) = z % 10*. Výsledkem (indexem) je zbytek po dělení deseti. Do této funkce vstupuje klíč. Jelikož se klíč skládá z více znaků, vstupuje do funkce pro jednoduchost pouze první znak klíče. Pro převod znaku na ASCII kód byla využita funkce *ord ()*. Výstupem je číslo od 0 do 9, které odpovídá indexu koše, do kterého bude prvek vložen.

Při vkládání dvojice klíč-hodnota program nejprve vypočítá index pomocí výše zmíněné funkce. Následně se prvek uloží jako nový seznam (list dvojice klíč, hodnota) do příslušného koše pomocí metody *append ()*, čímž v koši vzniká řetězec.

2. Čtení a validace vstupu

Program načítá data ze vstupního textového souboru řádek po řádku. Pro kontrolu případných chyb je každý řádek očíslován, aby uživatel mohl rychleji zkontrolovat, který řádek se jeví být problémový. Posléze je řádek očištěn od neviditelných znaků pomocí metody *strip ()*. Platné jsou pouze řádky, u nichž je klíč od hodnoty řádně rozdělen oddělovačem „;“, prázdné řádky jsou přeskočeny. Řádek je rozdělen oddělovačem pomocí metody *split (“;“)*. Následuje kontrola integrity. Pomocí logického operátoru *if* a funkce *len ()* je prověřován správný počet sloupců.
V případě, že se na řádku vyskytuje více sloupců, bere se v potaz první sloupec jako klíč a druhý sloupec jako hodnota. Ostatní případné sloupce jsou ignorovány a program vypíše varovnou zprávu s číslem problémového řádku. V případě, že se v řádku nachází pouze jeden sloupec, řádek je přeskočen a program taktéž vypíše varovnou zprávu.

3. Zpracování a formátování výstupu

Po zpracování všech validních řádků ze vstupního souboru následuje generování reportu. Tato část programu transformuje vnitřní reprezentaci dat do přehledného textového souboru.

Algoritmus postupně prochází hlavní pole *self.kose*. Pro každý koš pomocí funkce *len ()* zjistí počet dvojic. Posléze vytvoří seznam, do kterého jsou přidávány dvojice náležící příslušnému koši. Pro zpřehlednění jejich výpisu byla použita metoda *join ()*, která prvky z dvojice spojí znakem ‘,‘. Metoda *write* umožní vypsání obsahu příslušného koše do požadovaného souboru.

4. Výjimky

Pro práci se soubory byl využit bezpečný blok řešící výjimky *try-except*. V případě nenalezení souboru program vypíše chybové hlášení. Taktéž i při výskytu neočekávané chyby.

Vstupní a výstupní data:

Vstupním souborem je v tomto případě soubor s názvem „vstup_separate_chaining.txt“. Jedná se o textový soubor obsahující jména a počty bodů, které jsou od sebe odděleny středníkem ve formátu: „Jméno; počet bodů“. Na každém řádku je uvedena právě jedna dvojice. V této dvojici je „Jméno“ klíčem, který vstupuje do hashovací funkce a „počet bodů“ zastává hodnotu.

Výstupním souborem je textový soubor s názvem „vystup_sc_slovak.txt“. V tomto souboru se nachází 10 řádků. Každý z nich zastává obsah jednoho koše hashovací tabulky. U jednotlivých košů je uveden jejich index (např. „Koš 0“) a počet prvků v koši. Dvojtečkou je pak oddělen samotný obsah koše, kde jsou vypsány všechny dvojice, které do daného koše přísluší.

Problematická místa a možná vylepšení:

Problematická v tomto případě může být samotná hashovací funkce, která je v mém programu poměrně slabá. Vzhledem k tomu, že do ní vstupuje pouze první znak klíče, dochází k nerovnoměrnému zaplnění jednotlivých košů. V mém případě jsem jako klíč zvolil jméno, což není ideální, jelikož jména se stejnými počátečními písmeny nejsou rovnoměrně rozložena. Jako možné vylepšení se zde nabízí matematická operace ASCII kódů všech znaků ve jméně, jejíž výsledek by vstupoval do hashovací funkce.

Další místo hodné pozornosti je samotná tabulka. Její fixní velikost 10 košů umožňuje efektivní práci s malým množstvím prvků. Při vyšším počtu dvojic by došlo k přílišnému zvětšení řetězců, které by již nebylo efektivní pro vyhledávání v databázi. Řešením tohoto problému může být implementace tzv. Resizing. V případě naplnění tabulky nad určitou mez by došlo k navýšení počtu košů a novému přepočítání všech prvků pro nové rozřazení.

Při validaci dále chybí kontrola duplicit. V případě, že se v souboru budou některé prvky opakovat vícekrát (stejné jméno, stejný počet bodů), se v mém programu všechny duplicity zřetězí do stejného koše. Krok, řešící tento problém by mohl být

kód, který by prošel cílový koš se záměrem zjištění, zdali se v něm tento prvek již nenachází.