* Zložitosti
* Uml diagramy
* Popis vyhľadávania a mazania duplicít s ukážkami kódu

**K-D vyhľadávací binárny strom**

Prerekvizitou vkladania inštancií žiadaného typu do K-D stromu je implementácia rozhrania „*IKdComparable*“ pre kľúč vkladaných dát. Typ kľúča tým deklaruje, že inštancia vkladaného typu je schopná byť porovnávaná s kľúčom už vložených dát v strome podľa špecifikovanej dimenzie. Ďalej sa na túto metódu porovnávania budeme referovať ako na „*komparátor*“. Určenie dimenzie znamená uvedenie celého čísla, ktoré môže nadobúdať hodnoty počnúc číslom 1 až po číslo celkovej najvyššej dimenzie kľúča. Napríklad, ak chceme ukladať do K-D stromu dáta s kľúčom, ktorý sa skladá z polohy X a polohy Y, pričom prvú dimenziu kľúča určuje poloha X a druhú dimenziu zas poloha Y, potom do metódy porovnania kľúčov môže byť pre dimenziu uvedené číslo z množiny {1,2}.

Operácie K-D stromu:

**Vloženie**

V K-D strome sa ukladajú dáta tak, že ich finálna pozícia sa určí postupným vetvením v strome – hľadaním miesta, kde uložiť dáta na základe hodnoty ich kľúča. V strome sa porovnávajú hodnoty kľúčov nasledovne:

1. Podľa aktuálnej úrovne porovnávaného prvku v strome sa určí dimenzia. Dimenzia sa určí ako (aktuálna hĺbka v strome MODULO počet dimenzií kľúča) + 1. Hĺbka v koreni nadobúda hodnotu 0, v synoch koreňa hodnotu 1 a pod.
2. Porovnajú sa všetky atribúty kľúča, ktoré určujú hodnotu aktuálnej dimenzie (zistenej v predcházajúcom bode) pre inštanciu dát, ktoré sa majú vložiť, s kľúčom inštancie aktuálne porovnávaného vrcholu v strome pomocou komparátora.
   1. Ak komparátor vráti výsledok, že vkladaný prvok s kľúčom má pre aktuálne porovnávanú dimenziu hodnotu kľúča menšiu alebo rovnú voči hodnote pre danú dimenziu prvku v strome, postupujeme - vetvíme v stromovej štruktúre smerom na ľavého syna.
   2. Pokiaľ je hodnota vkladaného kľúča v danej dimenzií väčšia ako existujúca v strome, postupujeme smerom do pravého podstromu.
3. Tento postup sa aplikuje, kým porovnávaný prvok zo stromovej štruktúry nemá syna pre podstrom, na ktorý by sme mali postupovať. Na toto miesto sa vloží prvok s kľúčom.
   1. Existuje ešte špeciálny prípad, kedy strom je prázdny. Vtedy sa vkladaný prvok iba označí ako koreň stromu.

Zložitosť operácie vkladania je pre n prvkov (bez degradácie stromu na zreťazený zoznam) logaritmická log2(n), pretože každým vetvením nemusíme porovnávať pravý alebo ľavý podstrom vďaka pravidlu, že hodnoty kľúča pre relevantnú dimenziu prvku v strome, ktoré sú menšie alebo rovné, budú vždy v ľavom podstrome a naopak, hodnoty kľúča väčšie pre danú dimeniziu, sa budú nachádzať v pravom podstrome.

Na obrázku 1 je znázornený kód vkladania do K-D stromu.



Obrázok 1 – vkladanie do K-D stromu

**Vyhľadávanie**

Strom podporuje 2 typy vyhľadávania, pričom prvý je súčasťou druhého:

1. prvky s rovnakou hodnotou sekundárneho kľúča – K-D strom dovoľuje vkladanie dát s rovnakou hodnotou sekundárneho kľúča. Preto, keď vyhľadávame podľa sekundárneho kľúča, musíme nájsť všetky. Pre účely nájdenia všetkých takých prvkov si vytvoríme pomocnú metódu „*findNodeWithKey*“ na nájdenie prvku s kľúčom, ktorý mal ako prvý zhodu so sekundárnym kľúčom. V tejto sa bude hľadať „*miesto*“ ako pri vkladaní prvku do stromu a to nasledovne:
   * Budeme hľadať cestu, po ktorej by musel kľúč putovať na svoje miesto pri vložení porovnávaním hľadaného kľúča s kľúčom aktuálne porovnávaného prvku stromu na základe dimenzie, podľa ktorej sa vetví na jeho úrovni. Ak je hodnota hľadaného kľúča v dimenzií menšia, musíme hľadať v pravom podstrome. Ak je hodnota väčšia, ideme do pravého podstromu. Ak sa však hodnota kľúča pre danú dimenziu rovná hodnote kľúča aktuálne porovnávaného prvku, porovnáme hodnoty kľúčov pre všetky dimenzie pomocou metódy „*hasSameKey*“. Ak sa nezhodujú, vetvíme doľava, ale ak sa našla zhoda, vrátime refenciu na prvok, ktorý má rovnakú hodnotu kľúča pre všetky dimenzie.



Obrázok 2 – metóda „findNodeWithKey“

* + Vyššie opísanú metódu budeme pri hľadaní všetkých duplicít využívať opakovane. Parametrom metódy „*findNodeWithKey*“ je vrchol podstromu, počnúc ktorým sa má začať vyhľadávanie. Na naprogramovanie vyhľadania duplicít využijeme nasledovnú skutočnosť. Keď do stromu vkladáme prvok s kľúčom, ktorého hodnota - pre všetky dimenzie - sa v strome už nachádza, potom tento prvok pôjde rovnakým vetvením smerom k listom stromu. Ak počas vetvenia v strome narazí na prvok s rovnakým kľúčom vo všetkých dimenziách, odhliadnúc od toho podľa akej dimenzie sa vetví v danej hĺbke, potom musí vždy pokračovať hľadanie svojho miesta v ľavom podstrome, pretože pre ľubovoľnú dimenziu je hodnota kľúčov rovnaká. Túto vlastnosť využijeme nasledovne. Pokúsime sa nájsť, počnúc koreňom, prvú zhodu s hľadaným kľúčom využitím metódy „*findNodeWithKey*“. Prvok, s ktorým sa našla zhoda si uložíme z návratovej hodnoty metódy do zoznamu nájdených duplicít. Následne opakujeme volanie metódy, ktorá začne svoje vyhľadávanie v ľavom synovi nájdenej duplicity. Toto volanie sa opakuje, kým nám metóda „findNodeWithKey“ vráti nejaký prvok. Dôležité je, aby sme metóde posúvali aj parameter hĺbky začiatočného vrcholu vyhľadávania, aby sme vedeli, podľa akej dimenzie máme vetviť vo vyhľadávaní.
  + Zložitosť tohto vyhľadávania pre n prvkov s duplicitami je logaritmická log2(n), pretože vždy kontrolujeme iba jeden z dvoch podstromov. Pri opakovanom volaní metódy „*findNodeWithKey*“ iba pokračujeme v prehľadávaní podstromu, nekontrolujeme žiadne vrcholy navyše. Metóda teda iba zaobaľuje blok kódu nájdenia najbližšej duplicity. 

Obrázok 3 – vyhľadávanie prvkov na základe hodnôt sekundárneho kľúča

1. unikátny prvok – nájdenie unikátneho prvku v K-D strome so sekundárnymi kľúčmi dosiahneme pridaním obmedzenia pre typ vkladaných dát tak, že musia realizovať rozhranie „ISame“, ktoré obsahuje metódu „isSame“. Na túto metódu sa ďalej budeme referencovať ako „overovateľ“. Typ vkladaných dát sa realizáciou zaväzuje, že vie vyhodnotiť na základe unikátneho atribútu či je inštancia tá istá na základe referencie na inštanciu alebo hodnoty unikátneho atribútu. So splnením tejto prerekvizity sa v K-D strome vyhľadajú všetky duplicity na základe sekundárneho kľúča a následna sa z týchto duplicít nájde konkrétny prvok hľadaných dát použitím overovateľa.
   * Zložitosť nájdenia unikátnych dát pre strom s n prvkami je log2(n) + k, kde číslo „*k*“ predstavuje počet nájdených duplicít sekundárneho kľúča, ktoré je nanajvýš potrebné skontrolovať pre nájdenie dát s jedinečným atribútom. Výsledná zložitosť vyhľadávania je O(log2(n)).



Obrázok 4 – vyhľadanie konkrétneho prvku na základe unikátnej hodnoty využitím overovateľa

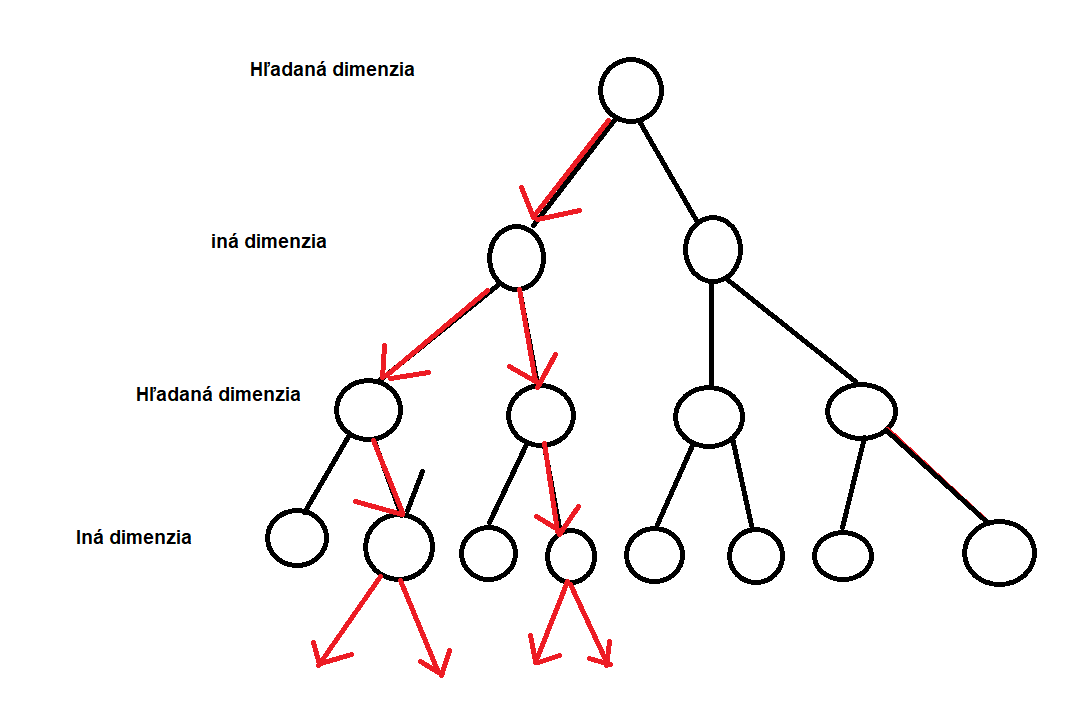
**Vymazanie**

Vymazanie prvku z K-D stromu sa principiálne skladá z dvoch častí.

1. Nájdenie prvku, ktorý sa má vymazať nájdený pomocou komparátora a overovateľa.
2. Vymazanie samotného prvku:
   1. Pokiaľ je prvok listom (nemá žiadného syna), môžeme ho jednoducho zo stromu odobrať
   2. Pokiaľ má prvok syna v ľavom alebo pravom podstrome, musíme odstraňovaný prvok nahradiť nejakým prvkom z podstromu tak, aby sa nenarušila štruktúra stromu, t.j. aby po odstránení prvku bolo stále možné vyhľadať ľubovoľný prvok v pravom aj ľavom podstrome.

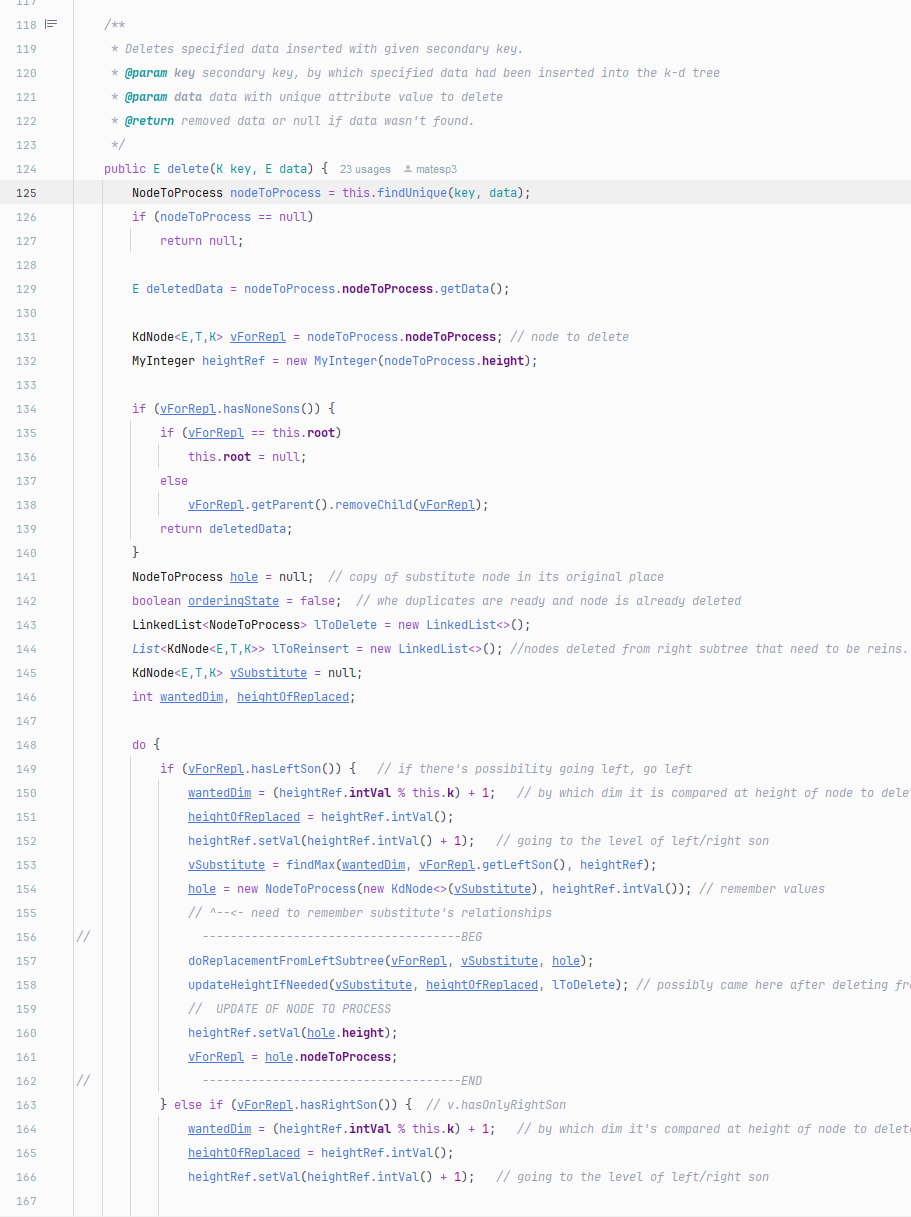
Vymazanie z K-D stromu, ktorý uchováva duplicity, je špecifické na základe toho či vymazávaný prvok nahrádzame prvkom z ľavého alebo pravého podstromu.

1. Z ľavého podstromu odstraňujeme tak, že si nájdeme prvok s maximálnou hodnotou pre dimenziu, pre ktorú hľadáme náhradníka. Na úrovniach, kde sa vetví podľa hľadanej dimenzie, vždy vetvíme iba vpravo, keďže maximum sa môže nachádzať len v pravom podstrome. Ak sme na úrovni, kde sa vetví podľa inej ako hľadanej dimenzie, musíme prehľadať obe vetvy.



Obrázok 5 – znázornenie hľadania maxima v ľavom podstrome pre danú dimenziu

Keď nájdeme prvok s maximom, vymeníme ho s prvkom, ktorý sa má vymazať. Ak vznikla diera na mieste náhradníka, musíme postup opakovať výmeny z ľavého alebo pravého podstromu až kým sa s dierou nedostaneme do listu.



Obrázok 6 - hľadanie náhradníka z ľavého podstromu

1. Z pravého podstromu sa nájde minimum obdobne ako hľadanie maxima v ľavom podstrome, akurát namiesto vetvenia vpravo pre hľadanú dimenziu, sa vetví vľavo. Tu však pri hľadaní minima si uchovávame zoznam všetkých duplicít, ktoré sme na ceste k listu našli. Prvý z duplicít dáme na miesto odstraňovaného a ostatné je potrebné postupne zo stromu odobrať a opätovne vložiť po tom, ako všetky duplicity zo stromu odstránime. Pre každú duplicitu, pokiaľ je možné, robíme výmeny z ľavého podstromu, lebo týmito výmenami nemôžeme narušiť štruktúru stromu a vyhýbame sa tým potenciálnemu riešeniu znovu vloženia duplicít do stromu, ktoré nemôžu byť v pravom podstrome po vložení jedného z nich za odstraňovaný prvok. Ak musíme opäť z pravého podstromu hľadať minimum, nové duplicity iného minima nájdené počas cesty vložíme do zoznamu prvkov na vymazanie a zoznamu opätovné vloženie.



Obrázok 7 – hľadanie náhradníka z pravého podstromu

1. Keď sa pri výmenách dostaneme do listu stromu, musíme sa pozrieť či v zozname prvkov, ktoré ešte musíme znovu vložiť do stromu, sa nachádzajú nejaké nespracované prvky. Ak áno, vyberieme jeden zo zoznamu prvkov na vymazanie a obdobne ich ľavými, resp. pravými výmenami odstraňujeme zo stromu až kým zoznam nie je prázdny. Následne sa všetky duplicity opätovne vložia do stromu na korektné miesto a tým sa mazanie končí.



Obrázok 8 – postupné odstraňovanie duplikátov, ktoré musia byť znovu vložené do K-D stromu

Zložitosť odstránenia prvku z K-D stromu sa skladá z:

1. Nájdenia unikátneho prvku na vymazanie spomedzi všetkých duplikátov sekundárneho kľúča. Teda pre n prvkov stromu a k duplikátov je zložitosť vyhľadania unikátneho prvku log2(n) + k.
2. Nájdenia extrému pre dimenziu, na úrovni, z ktorej sa ide prvok odstrániť. Nájdenie maxima, resp. minima znamená prehľadať ľavý, resp. pravý podstrom, ak sme na úrovni hľadanej dimenzie. Pokiaľ sme na úrovni, kde sa vetví podľa inej dimenzie, musíme prezrieť oba podstromu doteraz spomenutým spôsobom. Pre n prvkov v strome to znamená logaritmickú zložitosť log2(n) nájdenia extrému.
3. Po nájdení extrému vďaka referenciám vieme s O(1) zložitosťou vymeniť referencie odstraňovaného prvku s otcom a synami za nové refencie s náhradným vrcholom.
4. Tento postup potrebujeme aplikovať t-krát, kde t vyjadruje počet potrebných výmen, ktoré sú realizované po odstránení chceného prvku, aby sa nenarušila štruktúra K-D stromu.

Celková zložitosť odstránenia z K-D stromu pre n prvkov v strome je (log2(n) +k + t\*(1+log2(n))) = (k+t+(t+1)\*log2(n)), čo je logaritmická zložitosť O(log2(n)).

**Úprava**

Úpravu inštancie prvku, ktorá už existuje v K-D strome som neriešil na úrovni K-D stromu, ale až na klientskej úrovni používania štruktúry, pretože ide o kombináciu základných operácií. Dochádza totiž k dvom postupom na základe toho či upravujeme hodnotu kľúča alebo hodnotu z dátovej časti.

* Pokiaľ upravujeme hodnotu kľúča, musíme inštanciu zo stromu vymazať (logaritmická zložitosť), upraviť (jednotková zložitosť) a následne opäť vložiť do stromu (logaritmická zložitosť). Celková zložitosť bude teda logaritmická O(log2(n)) pre n prvkov.
* Ak chceme zmeniť iba položku/y z dátovej časti prvku stromu, stačí získať referenciu na konkrétny prvok zo stromu a cez ňu upraviť dátovú časť. Prvok sa teda najskôr nájde spolu so všetkými duplikátmi podľa sekundárneho kľúča v strome s n prvkami (logaritmická zložitosť), vyberie sa spomedzi „*k*“ duplicít a nastaví sa nová dátová hodnota. Zložitosť je zložená z log2(n) + k +1, čo je celkovo logaritmická zložitosť O(log2(n)).

Zhrnutie asymptotických zložitostí operácií na K-D strome:

|  |  |
| --- | --- |
| **Operácia** | **Zložitosť** |
| *Nájdenie všetkých prvkov so sekundárnym kľúčom* | Log2(n) |
| *Nájdenie unikátneho prvku* | Log2(n) |
| *Vloženie prvku* | Log2(n) |
| *Vymazanie prvku* | Log2(n) |
| *Úprava prvku* | Log2(n) |

Tabuľka 1 – výsledné zložitosti operácií K-D stromu