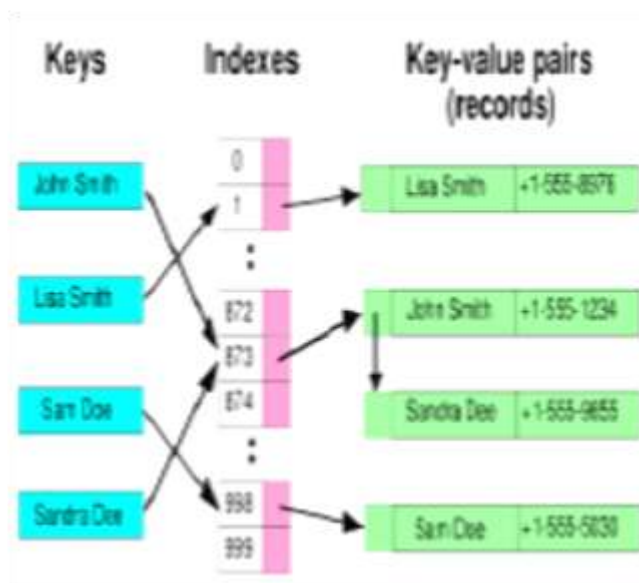


Keresés hasító táblázatokban. Láncolt listás és közvetlen címzéses hasító technika. Bináris fán alapuló keresés. BFák alkalmazása nagy adatbázisokban. Keresés, beszúrás, törlés, kiegyensúlyozás.

Egy hasító tábla egy olyan adatstruktúra, amelyik kulcsokat párosít össze értékekkel. Az elsődleges célja, hogy hatékonyan lehessen benne keresni: egy megadott kulcshoz hatékonyan találja meg a megfelelő értéket. A működés alapja, hogy a kulcsot egy hasító (hash) eljárás egy hash értéké alakítja, ami egy numerikus érték, és a tömbindexhez hasonlóan használja a kívánt tárolási pont azonosításához (ahol az érték tárolódik). A hasító táblák minden művelete (beillesztés, törlés, keresés) konstans idő alatt fut le, ami miatt még nagy táblák esetén is a karbantartás és az elérés hatékony tud lenni.

Ha két kulcsnak ugyanaz a hash értéke, akkor nem lehet őket egy helyen tárolni. Az ilyen ütközések elkerüléséhez valamilyen jó tárolási eljárást érdemes használni, ami megoldja ezeket a problémákat. Az ütközés problémája abból adódik, hogy egy adott, kulcsokat tartalmazó (K) elemeivel azonosított rekordok száma kisebb (jóval kisebb), mint a K számossága (azaz amennyi kulcs van). Ekkor K-t leképezzük egy alkalmas függvénnyel (ez a hash függvény) az ábrázolás alapját képező legkisebb tartományra (M). De mivel az M jóval kisebb, mint a K, ezért fel fog lépni kulcsütközés, azaz két nem egyező kulcsnak ugyanaz lesz a hash értéke. És itt jön a képbe a tárolási technika, aminek feladata ezt az ütközést feloldani.

A láncolt listás hasító technikában az adattárolás alapját a láncolt lista adja (a láncolt lista egyike a számítógép programozásban használatos legegyszerűbb adatszerkezeteknek. Olyan csomópontok, cellák sorozatából épül fel, amelyek tetszőleges számú és fajtájú adatmezőt, és egy vagy két hivatkozást tárolnak. A hivatkozás(ok) a lista következő (és előző) elemére mutat(nak)). A hasító tábla elemei a fejelemek, ezek rendre azokra a listákra mutatnak, amelyek az adott részre (a hasító tábla egy tárolási helye) leképzett kulcsokat tartalmazzák. Azaz minden egyes hasító tábla elem maga is egy láncolt lista, ami tartalmazza az indexet, és egy mutatót egy láncolt listára, amelyik láncolt listában megtalálható minden, az adott hash értékhez tartozó kulcs (és az értéke).



Elvileg szükséges lehetne rendezni a listát, de egy jó hash függvény eleve rövid listát készít, és azon rendezetlenül is gyors a keresés.

A közvetlen címzéses (vagy jellemzőbben nyílt címzéses) hasító technikánál a hasító táblák közvetlenül egy tömbben tárolják a rekordjaikat. Az ütközést próbálkozással oldja fel: ha egy kulcs

helye már lefoglalt, akkor szisztematikusan próbálkozik tovább, amíg nem talál egy szabad helyet. Többféle próbálkozási folyamat létezik:

- Lineáris próbálás: a próbák közötti intervallum kötött (jellemzően 1), azaz minden egyes sikertelen próba után a megadott intervallummal lép a függvény bal- vagy jobb irányban tovább a tömbön.
- Négyzetes próbálás: álvéletlen vagy pszeudo-véletlen próbálásnak is hívják, itt a lépésköz a hash értékkel arányosan növekszik. Tulajdonképpen az lépésközök itt is lineárisan növekednek, és az indexeket leíró függvény négyzetes.
- Kettős hash-elés: itt a lépésközöket egy másik hash függvény számolja ki.

A **bináris fák** olyan fa adatstruktúrák, ahol minden csomópontnak legfeljebb két leszármazottja lehet. Rendszerint ilyen fákból alakítanak ki bináris keresőfákat és bináris kupacokat. A bináris fákon való kereséseknek két jellemző algoritmusa van:

- Mélységi keresés (depth-first): az algoritmus minden lépésnél a legbaloldalibb, még nem bejárt csomópontot választja ki. Ha már nincs több lépés, akkor visszalép, és az utolsó elágazási pontban a következő, még nem bejárt alternatívát választja ki. A kiindulás történhet a gyökérpontból, vagy egy még nem bejárt csomópontból. Ha a kereső megtalálta a keresett csomópontot (adatot, kulcsot stb.), akkor leáll.
- Széltében keresés (breadth-first): az algoritmus mindig az adott kiindulási csúcsból, vagy az adott rétegből egy lépésben elérhető csúcsokat járja be. A gyökérpontot kiindulási pontnak tekintve a bináris fát olyan módon járja be, hogy mindig a gyökérhez legközelebb eső, még be nem járt csomópontot keresi fel.

A **B-fák** olyan speciális fa adatstruktúrák, amelyek az adatokat rendezett formában tárolja, és megengedi a keresést, beszúrást és törlést legrosszabb esetben is logaritmikus időben (adatok mennyiségének növekedésével a beillesztés és törlés műveletigénye logaritmikusan nő.). Egy csomópont egy előre meghatározott tartományban változó mennyiségű leszármazottat tartalmazhat. Beillesztés és törlés esetén a csomópontok száma változik, és hogy a leszármazott csomópontok száma a meghatározott tartományon belül maradjon, lehetséges csomópont egyesítés és szétválasztás is. Egy B-fát szokás jellemezni azzal, hogy egy csomópontnak hány leszármazottja lehet. Egy m B-fa kielégíti a következő feltételeket:

- Minden csomópontnak legfeljebb m leszármazottja lehet
- Minden csomópontnak (kivéve a gyökeret és a leveleket) legalább $m/2$ leszármazottja van
- A gyökérnek legalább 2 leszármazottja van
- Minden levél azonos szinten van, és nem hordoznak információt
- Egy nem-levél csomópont, aminek k leszármazottja van, $k-1$ kulcsot tárol

A B-fák nagyon alkalmasak adatbázisokban és fílerendszerekben való felhasználásra. Mivel maximalizált a leszármazottak száma egy csomópontban, ezért a fa magassága csökken, kevesebbszer kell kiegyensúlyozni, és nő a hatékonysága. Általában a maximális leszármazottak számát a diszk blokkméretéhez szokták igazítani, vagy más, hasonló mérethez.

Műveletek B-fákon:

- Keresés: a keresés a bináris fákkal analóg módon működik, az ott bemutatott kétféle algoritmus itt is használható.
- Beszúrás: minden művelet a levél elemeken történik. Lépései:
 - A fán kereséssel meg kell találni az a csomópontot, ahova az új elem beilleszthető

- Ha a csomópontnak kevesebb leszármazottja van, mint a maximális megengedett, akkor oda beszúrható az új elem, de figyelembe kell venni, hogy rendezettek maradjanak az elemek
- Más esetekben a levelet fel kell bontani két csomópontra:
 - Ki kell jelölni egy középelemet a régi és az új elemek közül
 - A mediánál kisebb elemek kerülnek az egyik új, a bal oldalon lévő levélre, a nagyobb elemek pedig a másik új csomópontra, a jobb oldalra. A medián mint elválasztó érték szerepel.
 - Az elválasztó érték hozzáadódik a szülő elemhez, ami abban a csomópontban okozhat szétválasztást, és így tovább, egészen a gyökér elemig (ha szükséges).
- Törlés: két féle megközelítés jellemző a törlésre:
 - kijelölni és törölni az elemet, majd újrastrukturálni a fát
 - végigmenni a fán egészen a kijelölt csomópontig, de még a csomópont elérése előtt újrastrukturálni a fát, hogy amikor törölve lesz az adott elem, utána már ne kelljen ezzel foglalkozni.

Két speciális eset lehet, amivel még törléskor foglalkozni kellhet:

- egy elem egy csomópontban elválasztó elemként szerepel: a bal oldali leszármazott legnagyobb és a jobboldali leszármazott legkisebb eleme közül valamelyiket ki kell jelölni, mint új elválasztót, és újra kell építeni az alatta lévő struktúrát
- egy elem törlésével a minimálisan szükségesnél kevesebb leszármazottja lesz egy elemnek: kiegyensúlyozás szükséges
- Kiegyensúlyozás: ha egy elem törlésével egy csomópontnak a kötelezően előírtnál kevesebb leszármazottja marad, akkor újra kell építeni a fát olyan módon, hogy mindenhol meglegyen a kötelező minimum. Bizonyos esetekben ez a művelet hiányt képez a szülő elemeknél, így az újraépítés eltarthat egészen a gyökérelemig is.