Otsimine

Paisksalvestusmeetod / paiskadresseerimine / räsimeetod.
(Hashing)

Paisksalvestusmeetod ehk paiskadresseerimine

Paiskadresseerimine *(hashing)* - Teatav meetod, millega otsivõti andmete salvestuse ja võtu eesmärgiga teisendatakse aadressiks (EVS-ISO/IEC 2382-7:2002).

Paisksalvestusmeetod (ka räsimeetod) on algoritm, mis paneb suvalise pikkusega andmehulga vastavusse fikseeritud pikkusega andmehulgale. Tavaliselt on vasteks täisarv, mida saab kasutada massiivi indeksina.

Paisksalvestusmeetod ehk paiskadresseerimine

Paiskmeetodit kasutatakse **otsimiseks**. Tüüpilised realisatsioonid andmestruktuuridena on näiteks sõnastikud jms vastendustüübid *(mapping type)*, andmebaasi indeksid jms.

Eeldus: igas kirjes on **unikaalne võti**. Salvestamine toimub massiivi.

Otseadresseerimine

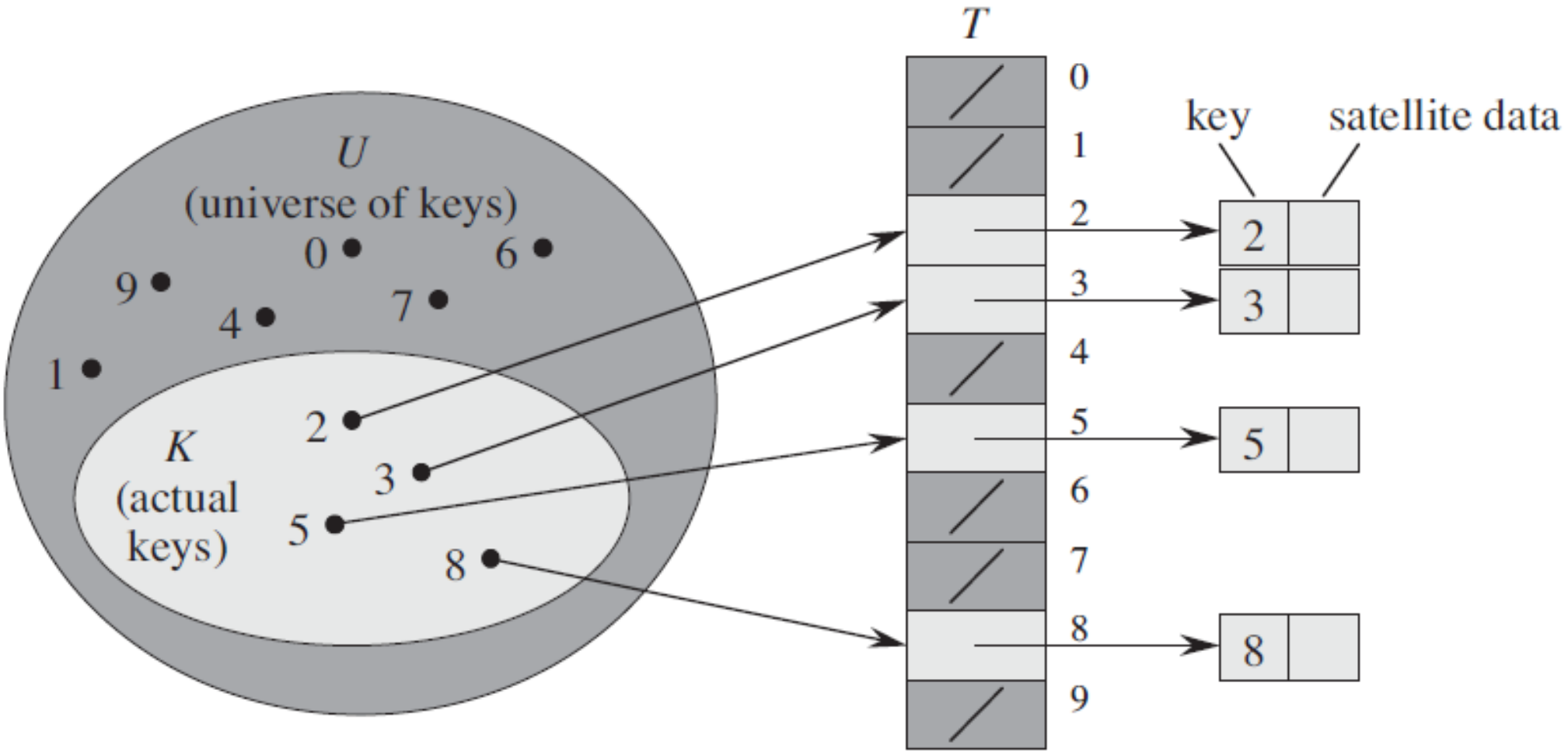
(Direct-adressing) – kui kirjete võtmed on täisarvud piiratud vahemikus, kasutatakse kirjete massiivi lisamisel indeksitena võtmeid endeid.

Kirje lisamisel pannakse massiivi indeksiks võti ning kirje salvestatakse vastavasse lahtrisse.

Kui kirjet otsitakse, vaadatakse lahtrisse, mille indeks võrdub võtme väärtusega. Kui kirje on seal, oli otsimine edukas.

Keerukusklass? Lisamisel ja otsimisel O(1) – sammude arv ei sõltu andmete hulgast ega kasva selle kasvamisel.

Otseadresseerimine



Joonis raamatust: Cormen, Leiserson, Rivest, Stein „Introduction to Algorithms", Third Edition

Võtmed paisk[e]tabelis

Kõik võtmed ei sobi otseadresseerimiseks.

Miks?

Paisk[e]funktsioon

Paisk[e]funktsioon (räsifunktsioon) *(hash function)* on algoritm, mis arvutab suvalisele väärtusele vasteks täisarvu nii, et see mahub etteantud vahemikku.

Funktsioon, mida kasutatakse teatud elemendi asukoha määramiseks elemendikogumis. (EVS-ISO/IEC 2382-7:2002)

Paiskmeetodi kontekstis on vahemikuks paisktabeli pikkus ehk leitud täisarv peab sobima tabeli (massiivi) indeksiks.

Seega on kaks hulka – võtmete hulk ja paisktabeli lahtrite hulk. Paiskfunktsioon `h(k)` vastendab võtme tabeli lahtrile, leides selleks sobiva indeksi.

Paiskeväärtus

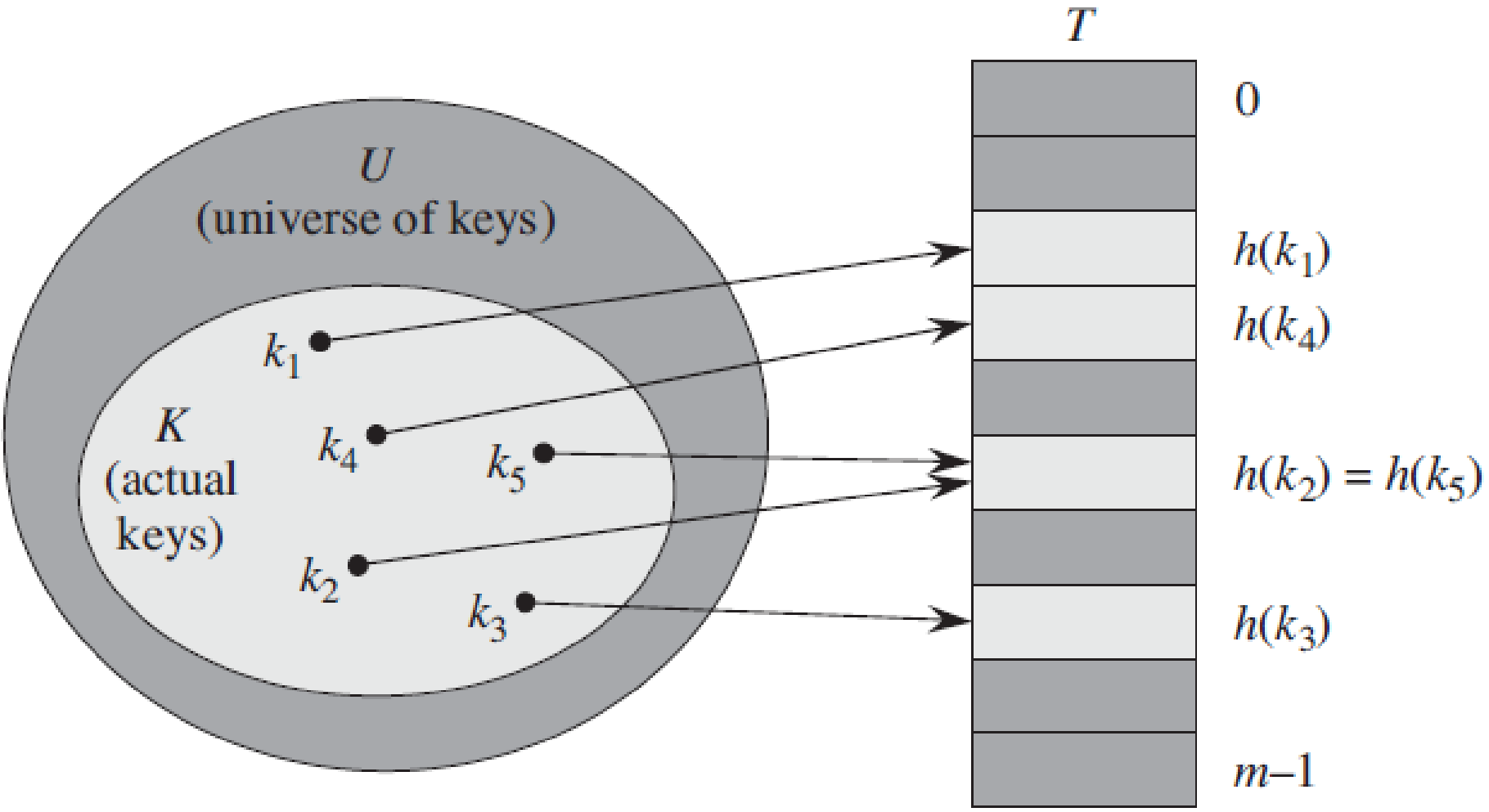
Paiskeväärtus *(hash value)* - paiskefunktsiooniga genereeritud arv elemendi asukoha näitamiseks mäluseadmes. (EVS-ISO/IEC 2382-7:2002)

Paiskeväärtuse leidmine võib toimuda ka kahe sammuga:

- leitakse (võtmele) arvuline väärtus;
- edasi rakendatakse arvulisele väärtusele paiskefunktsiooni.

Kui võtmed koosnevad sümbolitest, saab leida arvulise väärtuse sümbolite koode kasutades: `"ab" 97*128 + 98 = 12514`

Paiskfunktsiooni kasutamine



Joonis raamatust: Cormen, Leiserson, Rivest, Stein „Introduction to Algorithms", Third Edition

Paiskefunktsiooni valik

Hea paisekfunktsioon peab olema:

- **lihtne -** kiirelt ja kergelt arvutatav
- **üheselt määratud** – sama sisend peab alati andma sama väljundi
- **ühetaoline** *(uniform)* - jagama kirjed tabelisse võimalikult ühtlaselt (st arvutama indeksid, mis ühtlaselt jaotuvad).

Paiskefunktsiooni valik

Paiskefunktsiooni `h(k)` väärtused peavad paiknema vahemikus: `0 <= h(k) < M` kõigi lubatud võtmete `k` jaoks (`M` on tabeli pikkus).

Erinevat tüüpi võtmetele sobivad erinevad funktsioonid.

Jäägimeetod

Jäägimeetodi *(division method)* korral on paiskeväärtuseks jääk, mis tekib võtme täisarvulisel jagamisel tabeli pikkusega.

`h(k)=k mod M,`

kus `k` on võti ja `M` on paisktabeli pikkus.

`M`-i valik ei ole suvaline. Sobivad pigem algarvud.

Ei sobi arvusüsteemi alus, paarisarvud jms, mille puhul samasuguste jääkide (paiskväärtuste) tekkimise võimalus on suurem.

Korrutamise meetod

Korrutamise meetod *(multiplication method)* – paiskeväärtuse leidmiseks korrutatakse võti mingi irratsionaalarvuga `(0 < A < 1)`, täisosa lahutatakse. Järgi jääb arv vahemikus `0` kuni `1`. Leitud arv korrutatakse tabeli pikkusega `M` ja lõigatakse ära murdosa.

Irratsionaalarvuks sobib nt kuldlõige:

$T = (\sqrt{5}-1)/2 = 0,618033$

Paiskefunktsioon on (`[]` tähistab täisosa):

$h(k) = [M*(k*T - [k*T])]$

Kollisioon e paiskekonflikt

Paiskekonflikt e kollisioon *(collision)* - sama paiskeväärtuse ilmnemine kahe või mitme võtme puhul. (EVS-ISO/IEC 2382-7:2002)

See on olukord, kus paiskefunktsiooni rakendamisel kahele erinevale võtmele tekib sama paiskeväärtus.

Sisuliselt tähendab see vajadust paigutada kaks erinevat võtit massiivis samasse lahtrisse.

Kollisioonid on paratamatud, seega tuleb nende tekkimisel midagi ettevõtta.

Kollisioonid

Kollisioonid hakkavad tekkima tõenäosusega üle 0.5, kui tabelisse pikkusega `M` on paigutatud `sqrt(Pi*M/2)` kirjet.

Sünnipäevade paradoks: 23 juhuslikult valitud inimese hulgast on kahel inimesel ühel kuupäeval sünnipäev tõenäosusega 0.5 `(M=365)`.

Kollisioonide lahendamine

Kollisiooniohu vähendamiseks:

- tabel peaks olema suurem kui andmehulk (nt 1/3 .. 1/4 tabelist vaba);
- hea paiskefunktsiooni valik, mis sõltub ka tabelisse paigutatavate võtmete iseloomust.

Kollisiooni tekitanud kirjed paigutatakse:

- kollisiooniahelatesse;
- tabeli piiresse, leides üheselt määratud uue koha.

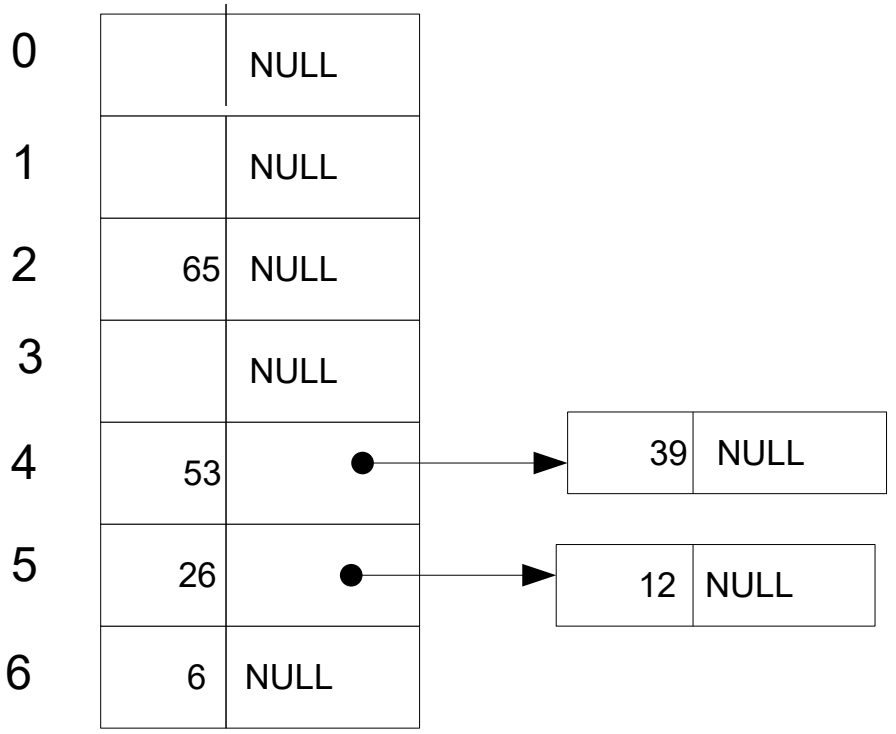
Näide (algus)

Paisketabel 7 lahtriga, jäägimeetod (võti mod 7)

Võti	Paiskeväärtus
26	5
53	4
12	5 (kollisioon)
65	2
39	4 (kollisioon)
6	6

Kollisiooniahelad

Eelmise slaidi võtmed paigutatuna paisketabelisse ja kollisioonid lahendatud ahelloendite abil:



Kollisiooniahelad.Selgitus

Paisketabeli lahter sisaldab võtit ja aadressivälja. Viimase külge on võimalik kinnitada ahelloend sama paiskeväärtusega võtmetest.

Teisisõnu (ehk pooleldi C-keeles): massiivi iga element on `struct`, nagu ühe viidaga ahelloendis. Aadressivälja külge saab "aheldada" järgmise kirje, mis on samasugune `struct`.

Tabelis võib olla kas ainult aadress (massiiv koosneb vaid viitadest) või lisaks esimene võti (massiivi elemendiks on `struct`).

Otsimine

Otsimine võtme `k` järgi:

- Arvuta välja paiskeväärtus `h(k)`.
- Kas võti `k` on tabelis kohal `t[h(k)]`?
- Kui lahter on tühi, siis otsimine on ebaedukas.
- Kui lahtris on `k`-st erinev võti, siis otsi vastavast kollisiooniahelast.
- Kui `k`-d ei leitud ka ahelloendist, siis ebaedukas otsimine.
- Kui võti `k` oli lahtris `t[h(k)]` või ahelloendis, siis edukas otsimine.

Lisamine

Võtme `k` lisamine (eeldusel, et kirjet võtmega `k` tabelis ei ole.)

- arvuta paiskeväärtus `h(k);`
- kui tabelis vastava indeksiga lahter on tühi, siis lisa võti (kirje) sinna;
- vastasel juhul lisa võti (kirje) kollisiooniahelasse.

Vaba paiskesalvestus

Vaba paiskesalvestus *(open hashing)* – kõik kirjed (võtmed) tuleb mahutada tabelisse.

Iga võtme jaoks kirjeldatakse lahtriaadresside ehk indeksite jada, kuhu proovitakse võtit paigutada, seni kuni leitakse vaba koht.

Aadresside järgnevust nimetatakse **sondeerimise järjekorraks** *(probing sequence)*.

Paiskefunktsioonile lisandub teine argument - sondeerimise järjekorranumber: `h(k,0), h(k,1), ...`

Üldine põhimõte

Võtme lõplik asukoht tabelis sõltub võtme paiskeväärtusest, sellest, mitmedat korda võtit tabelisse paigutada proovitakse ja võtmest endast.

Funktsiooniga `s(j,k)` kirjeldatakse sondeerimise järjekord. Argumentideks on võti `k` ja proovimise jrk number `j`. Viimane muutub `0..m-1`.

Sondeerimiste jada (indeksid, kuhu andmeid paigutada) arvutatakse välja valemiga:

```
(h(k)-s(j,k)) % m
```

Otsimine võtme k järgi

Alusta lahtriaadressist `i = h(k)`.

Otsi seni kuni leiad `k` või kuni lahter `t[i]` on vaba.

- Uus `i` arvuta valemist `i=(h(k)-s(j,k)) % m` nii, et `j` kasvab `0..m-1`.
- Kui `t[i]` sisaldab otsitavat võtit, on otsimine edukas ja see lõpetatakse.
- Kui `t[i]` on vaba, on otsimine ebaedukas ja see lõpetatakse.

Lisamine võtmega k

Eeldame, et võtit `k` tabelis ei ole.

- Alusta aadressist `i = h(k)`.
- Jätka lahtriaadresside leidmist valemiga

`i=(h(k)-s(j,k)) % m`

kuni lahter aadressiga `i` on vaba või kustutatud.

- Lisa sinna uus võti `k` ja andmed.

Kustutamine võtme K järgi

- Otsi võtmega `k`.
- Kui otsimine oli edukas märgista `t[i]` kustutatuks.

NB! Kustutamisel ei tohi kaotada infot selle kohta, et siin lahtris varem võti paiknes. Miks?

Lineaarne sondeerimine

Lineaarse sondeerimise *(linear probing)* puhul on sondeerimisfunktsioon:

`s(j,k) = j`

Kus `j = 0, 1, ..., m-1`

Meetodi puuduseks on lineaarne klasterdumine – tekivad pikad hõivatud piirkonnad, mille pikenemise tõenäosus kasvab.

Seetõttu kasvab rohkem täidetud tabelis ka operatsioonideks kuluv aeg, sest pikeneb sondeerimise jada.

Näide

Tabeli iga rida näitab tabeli seisu peale järjekordse võtme lisamist (võtmed ja paiskeväärtused on slaidil 16)

Indeks / **Lisatav võti**	**0**	**1**	**2**	**3**	**4**	**5**	**6**
26						26	
53					53	26	
12				12	53	26	
65			65	12	53	26	
39		39	65	12	53	26	
6		39	65	12	53	26	6

Hinnang

Sammude arv esimese vaba koha leidmiseks sõltub suuresti tabeli täidetusest.

Vastavalt R. Sedgewick'i raamatule *“Algorithms in C”* on sammude (katse) arvud võtme lisamiseks lineaarsel sondeerimisel järgmised. Tulemused on leitud statistilisi meetodeid kasutades.

Tabeli täidetus:	**'1/2**	**2/3**	**'3/4**	**9/10**
Edukas otsing	1,5	2,0	3,0	5,5
Edutu otsing	2,5	5,0	8,5	55,5

Ruutsondeerimine

Ruutsondeeriminel *(quatratic probing)* on indeksite jada järgmine:

`h(k),h(k)+1,h(k)-1,h(k)+4,h(k)-4,...`

st funktsioon on $s(j,k)=([j/2])^2(-1)^j$

$(-1)^j$-ga korrutamise abil saavutatakse + ja – märkide vaheldumine, `[]` tähistavad täisosa.

On parem lineaarsest sondeerimisest, tekib sekundaarne klasterdumine, kuna samale paiskeväärtusele on sondeerimiste jada samasugune.

Näide

Paisktabel slaidil 16 toodud võtmetest (lõppseis):

Indeks:	0	1	2	3	4	5	6
Võti:	6		65	34	53	26	12

Topelt paiskesalvestus

Topelt paiskesalvestusel *(double hashing)* määratakse lisaks teine paiskefunktsioon `h'(k)`, mida rakendatakse samale võtmele.

Tekkiv sondeerimiste jada on järgmine:

```
h(k),
(h(k)-1*h'(k)) % m,
(h(k)-2*h'(k)) % m,
(h(k)-3*h'(k)) % m, ...
```

Teine paiskefunktsioon

Teises paiskefunktsioonis sobib samuti jäägi leidmine, kuid `m`-i on vähendatud 1 või 2 võrra:

Näiteks:

```
h'(k) = 1 + (k % (m-2))
```

Ühe liitmine on vajalik tulemuse 0 vältimiseks.

Sondeerimisfunktsioonis korrutatakse teise paiskefunktsiooni väärtus proovimiste arvuga:

```
s(j,k) = j*h'(k)
```

Näide

Võtmed slaidilt 16 paigutatakse tabelisse, näha on tabeli seis peale iga võtme lisamist.

Parempoolses tulbas on uue paiskeväärtuse rehkendus (võtmele 6 arvutatakse seda 3 korda).

Teises paiskefunktsioonis leitakse jääk võtme jagamisel `m-2`-ga (ehk 5-ga).

Näide jätkub

Indeks lisatav võti	0	1	2	3	4	5	6	
26						26		26 mod 7=5
53					53	26		53 mod 7=4
12			12		53	26		(5-(1+12 mod 5)) mod 7=2
65		65	12		53	26		(2-(1+65 mod 5)) mod 7=1
39		65	12		53	26	39	(4-(1+39 mod 5)) mod 7=6
6	6	65	12		53	26	39	(6-(1+6 mod 5)) mod 7=4 (6-2*(1+6 mod 5)) mod 7=2 (6-3*(1+6 mod 5)) mod 7=0

Hinnang

Vastavalt R. Sedgewick'i raamatule *“Algorithms in C”* on topelt paiskesalvestusel sammude (proovimiste) arvud võtme lisamisel järgmised. Tulemused on leitud statistilisi meetodeid kasutades.

Võrreldes lineaarse sondeerimisega on see efektiivsem ka rohkem täidetud tabeli pihul.

Tabeli täidetus:	**'1/2**	**2/3**	**'3/4**	**9/10**
Edukas otsing	1,4	1,6	1,8	2,6
Edutu otsing	1,5	2,0	3,0	5,5

Paiskadresseerimise keerukus

Üldiselt paigutatakse paisadresseerimise meetod **konstantsesse keerukusklassi O(1)**. Sest võtmete arvu suurenemisel ei ole otsest seost sondeerimiste arvu kasvuga. Eeldusel, et tabel ei ole liiga täis.

See ei tähenda, et kõik operatsioonid oleksid tehtavad sõna otseses mõttes ühe sammuga.

Halvimal juhul saame aga lineaarse keerukuse O(N). Seda juhul, kui kõigile võtmetele arvutatakse sama paiskeväärtus.

Paisketabeli suurendamine

Kui tabel saab "liiga täis", tuleb tabelit suurendada. Tehakse tavaliselt poole suuremaks.

Kirjed paisatakse uude tabelisse *(rehashing).*

"Liiga täis" on suhteline mõiste. Sõltub kollisioonide lahendamise strateegiast:

Kollisiooniahelate puhul suurendatakse siis, kui võtmeid on sama palju kui tabelis lahtreid.

Vabal paiskesalvestusel aga sõltuvalt sondeerimise liigist, nt siis, kui tabeli täituvus (kirjete arv / pikkus) on 0,7.

Paiskadresseerimise kasutamine

Paisketabel on andmestruktuur, mis on aluseks teatud andmetüüpide realiseerimisel. Nt vastendustüübid (nn *mapppig type*), kus eesmärk on võtme järgi otsimine *(dictionary, associative memory, associative array)*.

Kompilaatorite poolt moodustatavad nimede (identifikaatorite) tabelid.

Andmebaaside indeksid.

Kui võtmed on nö hõredad ehk neid on vähe, kuid nende suuruste vahemik on suur.

Paiskadresseerimine vs otsingupuu

Mõlemad meetodid on mõeldud otsimiseks.

Reeglina on paiskemeetod parem / kiirem, eeldusel, et võtmed on lihtsamat andmetüüpi ja nendele on hea arvutada paiskeväärtust.

Kui võtmete arv ei ole ennustatav on puu parem oma dünaamilisuse tõttu.

Puu on parem, kui eeldada sorteeritust: sort jada, min, maks, naabervõtmed. Paisketabel nende leidmist ei võimalda. Tabeli saab küll väljastada, kuid see ei anna midagi.

Keerukusklasside võrdlus

Paiskemeetod

- Lisamine O(1)
- Kustutamine O(1)
- Otsimine O(1)
- Min O(N)
- Maks O(N)
- Sorted O(N log N)

Otsingupuu

- Lisamine O(log N)
- Kustutamine O(log N)
- Otsimine O(log N)
- Min O(log N)
- Maks O(log N)
- Sorted O(N)