Otsimine

Paisksalvestusmeetod / paiskadresseerimine / räsimeetod.
(Hashing)

Paisksalvestusmeetod

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

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

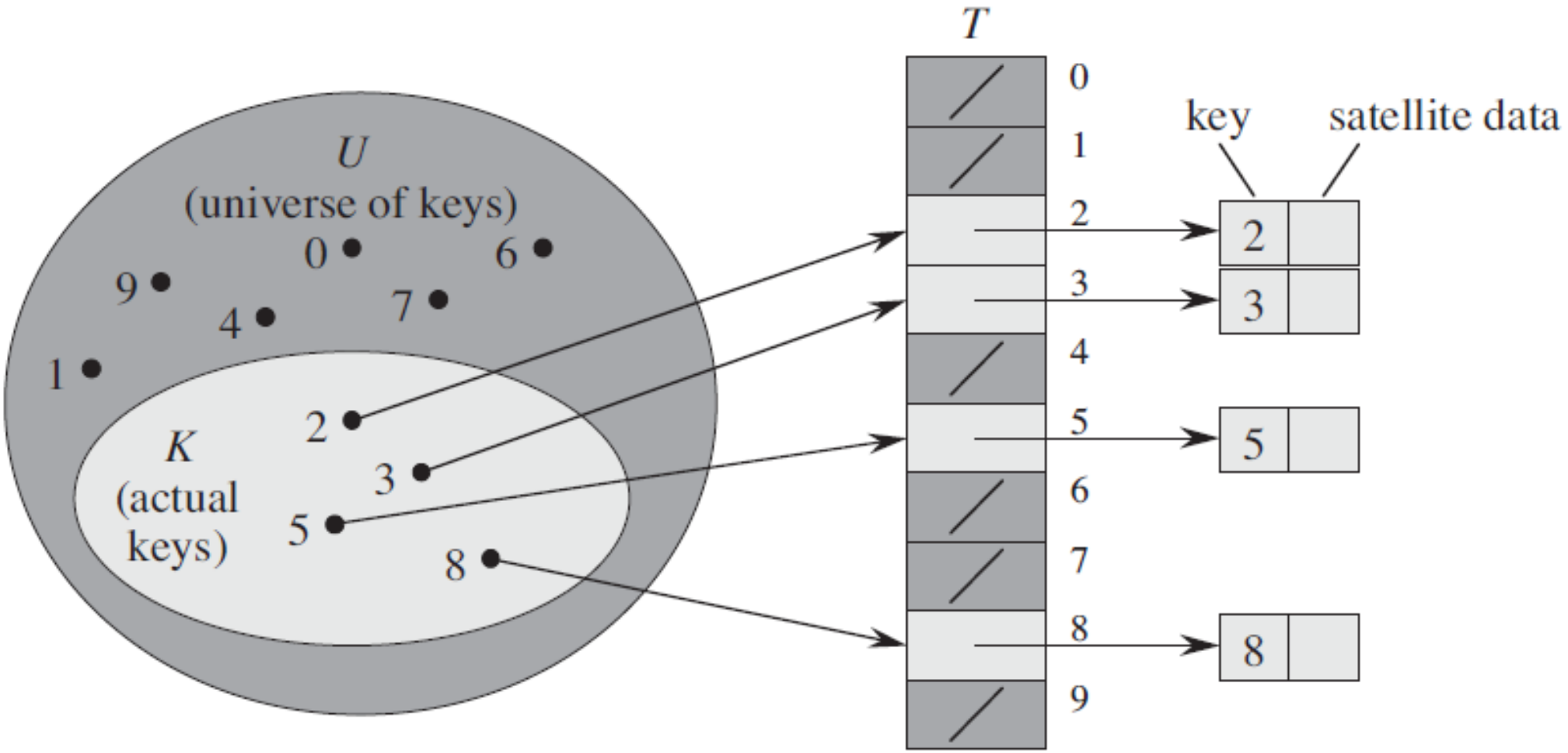
(Direct-adressing) – kui kirjete võtmed on täisarvud piiratud vahemikus, kasutatakse kirjete massiivi lisamisel indeksitena võtmeid endeid.

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?

Paiskfunktsioon

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

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.

Paiskväärtus

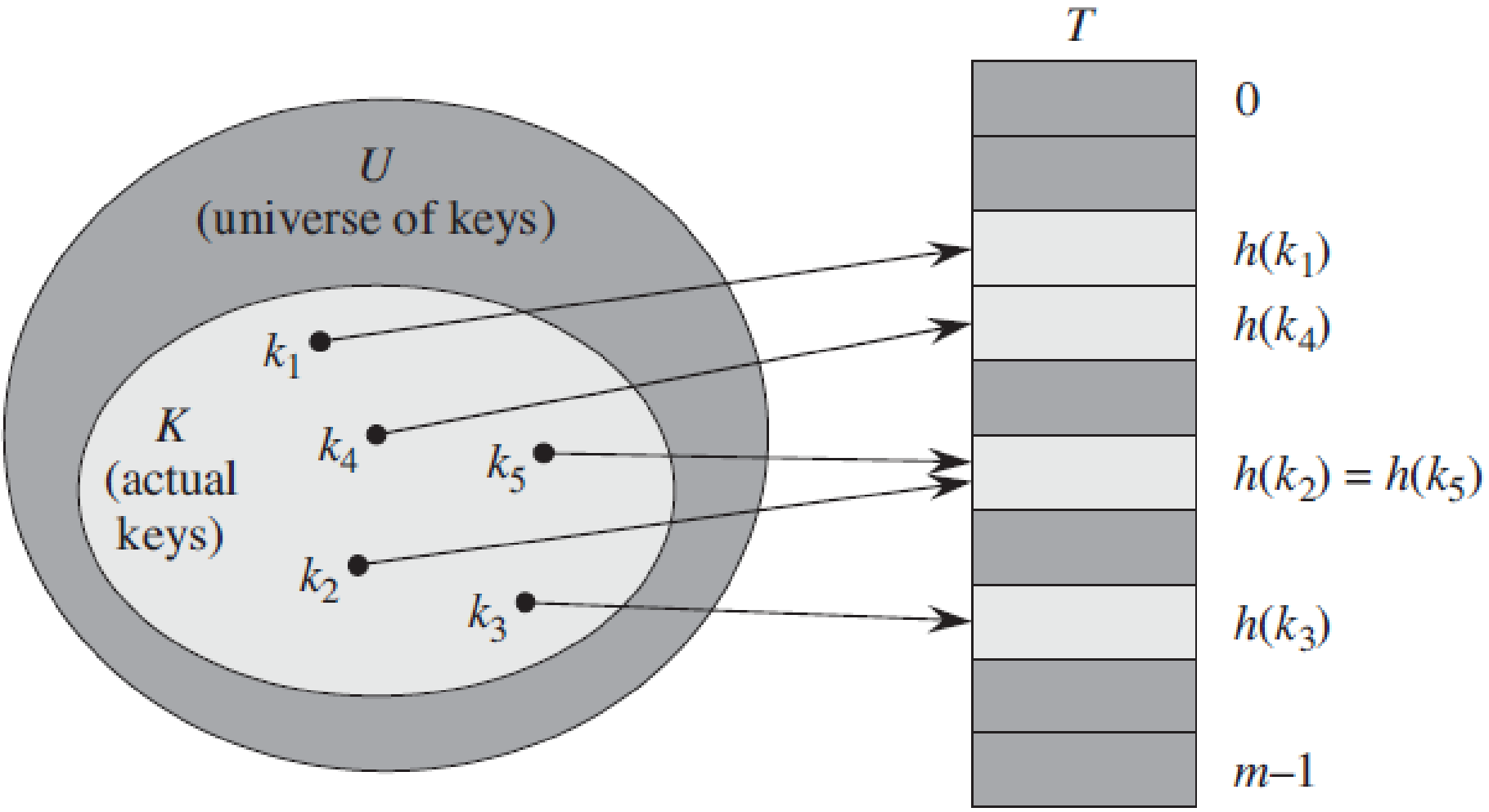
Leitud indeksit (aadressi) nimetatakse **paiskväärtuseks** *(hash value)*.

Paiskväärtuse leidmine võib toimuda ka kahe sammuga:

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

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

Paiskfunktsiooni valik

Hea paiskfunktsioon 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).

Paiskfunktsiooni valik

Paiskfunktsiooni `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)* puhul on paiskväärtuseks 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)* – paiskväärtuse 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:

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

Paiskfunktsioon on (`[]` tähistab täisosa):

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

Kollisioonid

Kollisioon *(collision)* ehk **põrge** on olukord, kus paiskfunktsiooni rakendamisel kahele erinevale võtmele tekib sama paiskväärtus.

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 paiskfunktsiooni valik, mis sõltub ka tabelisse paigutatavate võtmete iseloomust.

Kollisiooni tekitanud kirjed paigutatakse:

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

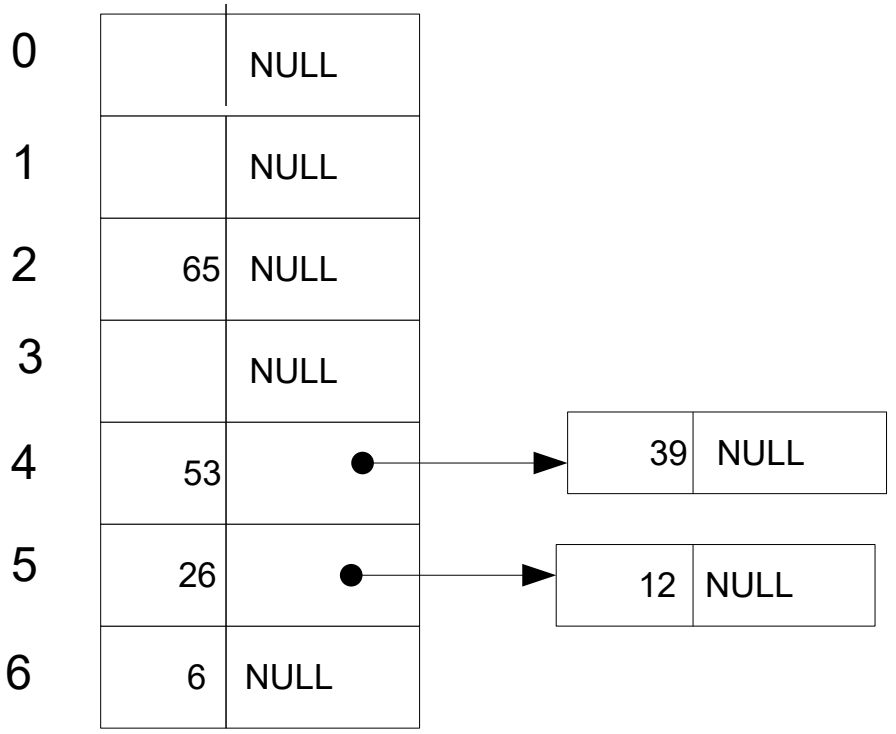
Näide (algus)

Paisktabel 7 lahtriga, jäägimeetod (võti mod 7)

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

Kollisiooniahelad

Kui eelmisel slaidil toodud võtmed paigutada paisktabelisse ja lahendada kollisioonid ahelate abil, saame järgmise "pildi".



Kollisiooniahelad.Selgitus

Paisktabeli lahter sisaldab võtit ja aadressivälja. Viimase külge on võimalik kinnitada ahel sama paiskväärtust omavatest võtmetest.

Teisisõnu (ehk pooleldi C-keeles): massiivi iga element on `struct`, nagu ühe viidaga ahelas. 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 paisväärtus `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 ahelast, siis ebaedukas otsimine.
- Kui võti `k` oli lahtris `t[h(k)]` või ahelas, siis edukas otsimine.

Lisamine

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

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

Vaba paisksalvestus

Vaba paisksalvestus *(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)*.

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

Üldine põhimõte

Võtme lõplik asukoht tabelis sõltub võtme paiskväärtusest, 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 paiskväärtused 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 paiskväärtusele 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 paisksalvestus

Topelt paisksalvestusel *(double hashing)* määratakse lisaks teine paiskfunktsioon `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 paiskfunktsioon

Teises paiskfunktsioonis 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 paiskfunktsiooni 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 paiskväärtuse rehkendus (võtmele 6 arvutatakse seda 3 korda).

Teises paiskfunktsioonis leitakse jääk võtme jagamisel `m-2`-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 paisksalvestusel 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

Paisksalvestuse keerukus

Üldiselt paigutatakse paisksalvestuse 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 paiskväärtus.

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

Paiskmeetodi kasutamine

Paisktabel 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.

Paiskmeetod / otsingupuu

Mõlemad meetodid on mõeldud otsimiseks.

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

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

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

Keerukusklasside võrdlus

Paiskmeetod

- 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)