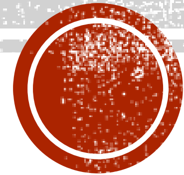
PRINCIPLES OF DATA ORGANISATION

Linear hashing extended edition



MOTIVATION

- Key, pointer pairs ~ index
- Linear hashing
- Issues with page utilization
  - Order of splitting is pre-defined by a condition which may not correspond to the data
- Optimizations:
  - Recursive linear hashing
  - Expansion techniques
    - Linear hashing with partial expansion (LHPE)
    - LHPE-RL
    - Spiral storage (not covered this semester)

RECURSIVE LINEAR HASHING

- Ramamohanarao & Sacks-Davis, 1984
- Employs recursive overflow handling
  - The overflow space is shared for overflow values
  - The overflow space is managed as a dynamically hashed file (i.e., linear hashing)
  - Pages in overflow areas may themselves be overflown → multiple levels of dynamic files
- The overflows are not explicitly linked from the primary page
  - We have a separate data structure
- At each overflow level ***i* = *1, 2, …*** there is also a linear hash file having:
  - depth d_i
  - split pointer p_i
  - number of pages at level $i = s_i + p_i$, where $s_i = 2^{d_i}$

RECURSIVE LINEAR HASHING – EXAMPLE

- When a record overflows, no overflow page is created but the record is inserted into the 2-nd level (and possibly recursively into 3-rd and so on)
- Worst case: number of accesses to the secondary memory = number of overflow levels + 1

p_0

0 … s_0-1

p_1

0 … s_1-1

1.overflow level

p_2

0 … s_2-1

2. overflow level

RECURSIVE LINEAR HASHING – SPLITTING

- Splitting of a page includes collecting of all the relevant records from all the next levels
  - When a page at level ***i*** is split, overflown records are collected from levels ***i*+1**, ***i*+2**, …
    - We search for recors with the same pre/suffixes = it is fast
  - If the primary page still overflows, the overflown records are put back into the first (and possibly following) overflown level
  - Decision whether to split can be controlled by the same splitting policy as in the standard linear hashing
- It has been shown that usually 3 levels are sufficient
  - The blocking factor is expected to be high (splits and overflows are not frequent)

RECURSIVE LINEAR HASHING – ADDRESSING

- Similar to linear hashing, but every level has different split pointer and number of pages

```
ADDR GetAddres(KEY k, int *cnt_pages, int cnt_levels) {
 bool found = false;
 for (int level = 0; level < cnt_levels; level++) {
  d = floor(log(cnt_pages[level], 2));
  s = exp(2, d);
  p = cnt_pages % s;
  addr = h(k) % s;
  if (addr < p)
   addr = h(k) % exp(2, d + 1);
  if (search(addr, level, k)){
   found = true; break;
  }
 }
 if (found) return addr; else return NULL;
}
```

search() searches page ***addr*** in level ***level*** for record with a key ***k*** and return the status of the search

EXPANSION TECHNIQUES

- Dynamic hashing schemes have oscillatory performance
- Having a uniform hash function causes all the pages to be filled more or less at the same time
- During a short period, many of the pages overflow and split
  - Utilization goes to N% and then the next moment drops to about 0.5N %
  - During the splitting period the cost of insertion is considerably higher
  - If overflow management techniques are employed, when the utilization approaches 100%, the cost of insertions and fetches increases
- Techniques to smooth the expansion were developed
  - Uniform distribution ~ linear hashing with partial expansion
  - Non-uniform distribution ~ spiral storage (not covered this semester)

LINEAR HASHING WITH PARTIAL EXPANSION (LHPE)

- Larson, 1980
- Overflow chains close to the right end of the unsplit region get too long at the end of the expansion stage
- Recently split pages are underutilized
- Pages near the right end are overutilized

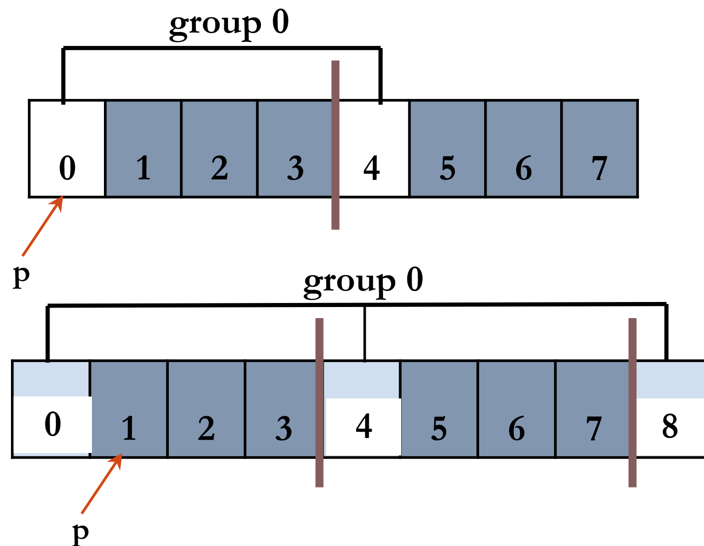
LHPE – Principal Idea

- In linear hashing:
  - Splitting after ***L*** insertions
  - We have ***s*** pages
  - So page ***s* – *1*** splits after ***sL*** insertions
- In LHPE we distinguish between partial and full expansion
  - During one full expansion the file expands in multiple partial stages
  - In each partial stage all the pages are split
- If the number of the partial expansion stages is 2, the pages ***s* – *1*** splits after ***sL* / *2*** insertions ~ shorter overflow chains

LHPE – SPLITTING A GROUP OF PAGES

d **= 3,** s **=** 2^d **= 8,** g **= 2 (pages in a group)**

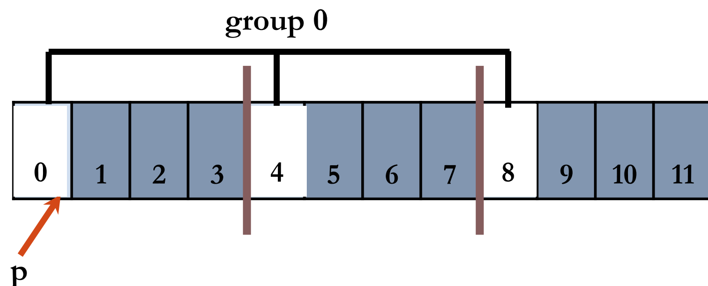
- We have 8 pages (0 .. 7)
- New page is added
  - page 8
- Records from the pages in a given group are spread across that group and the new page
  - pages 0, 4 and new page 8
- If b is the blocking factor and the pages are full then, utilization after the split operation is $2/3b$

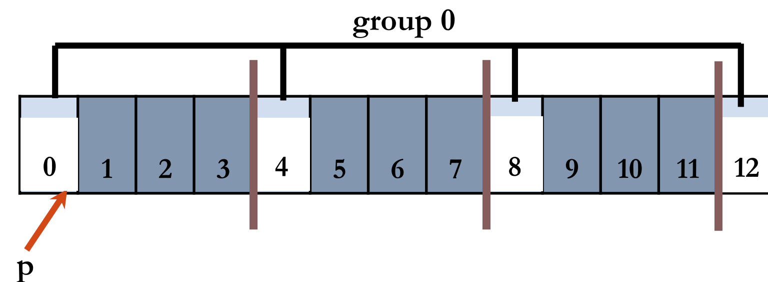


LHPE – PARTIAL EXPANSION

Situation after first partial expansion = splitting all groups

- We have visited each page in the original file
  - ***p*** passed through all the groups
- ***p*** returns to the page/group 0
  - Group 0 consists of pages 0,4,8
- We are halfway to the full doubling but we have already visited all the pages

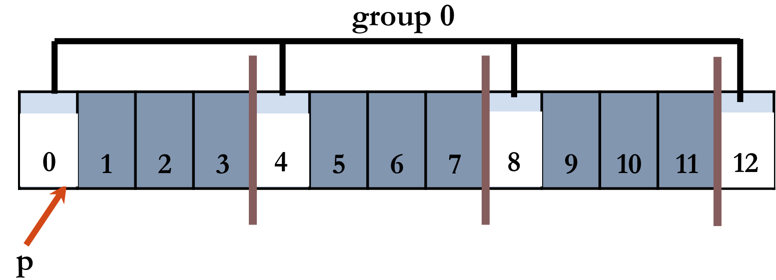




LHPE – PARTIAL EXPANSION

Second partial expansion

- Next page added will be 12
- If b is the blocking factor and the pages were full, then utilization after the split operation is about $3/4b$
- After this partial expansion, there will be 16 pages, the file will be doubled and one full expansion is over
  - Two partial expansions are over
- Size of each group will shrink to 2 again



LHPE – ADDRESSING

- To address a page we identify the respective group and compute the offset within that group
- When we are in d-th full expansion, the gap between pages of the same group is 2^{d-1}

  $$\boldsymbol{addr(k,d,n,p) = group + 2^{d-1} * offset}$$

  k … key, d … full expansion, n … partial expansion, p … split pointer
- Group determined as in linear hashing
- Offset determined by another given hash function mapping into the size of the group

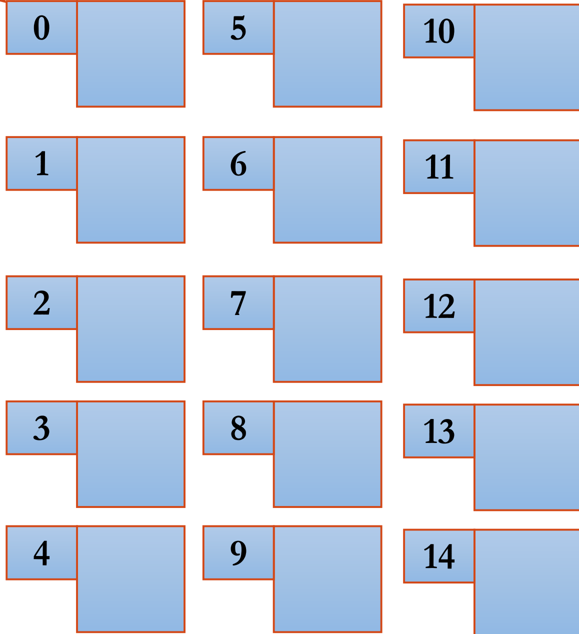
LHPE-RL

- Ramamohanarao & Lloyd, 1982
- Simplified version of LHPE
  - Partial expansion in LHPE corresponds to full expansion in LHPE-RL
  - Pages of the primary file (having $\boldsymbol{p_d}$ pages) at stage $\boldsymbol{d}$ are grouped into $\boldsymbol{s_d}=\boldsymbol{p_d}/\boldsymbol{g}$ groups (each having $\boldsymbol{g}$ pages)
  - When a predefined condition is met (e.g., after $\boldsymbol{L}$ insertions), a new page is inserted at the end of the primary file and records in pages in the group pointed to by the split pointer are redistributed between those pages and the new page (being the new member of the group)
  - When the last group is redistributed, the file is (virtually) reorganized (stage $\boldsymbol{d}+\mathbf{1}$) so that all the pages are again sorted into $\boldsymbol{s_{(d+1)}}=\boldsymbol{p_{(d+1)}}/\boldsymbol{g}$ pages ($\boldsymbol{p_{d+1}}=\lceil \boldsymbol{s_d}*(\boldsymbol{g}+\mathbf{1})/\boldsymbol{g}\rceil*\boldsymbol{g}$)

LHPE-RL – EXAMPLE

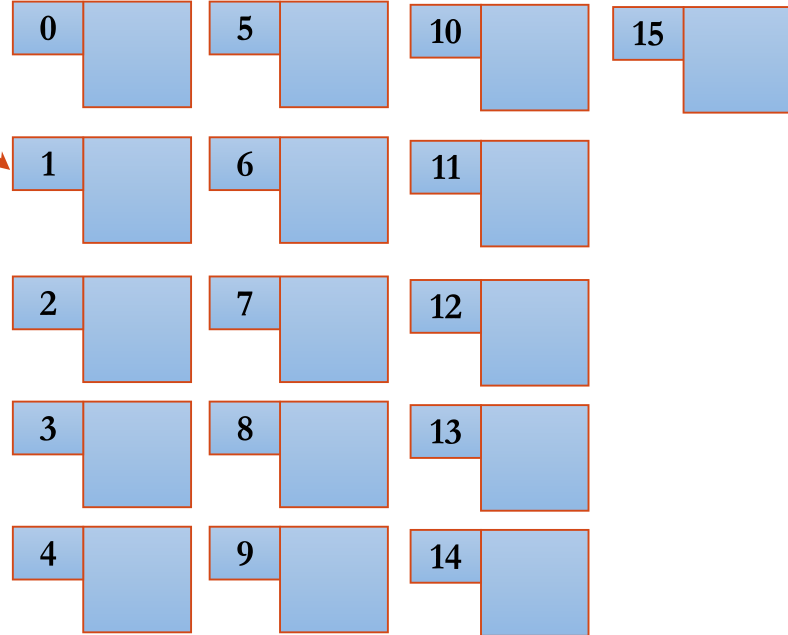
split
pointer

$s_d = 5$

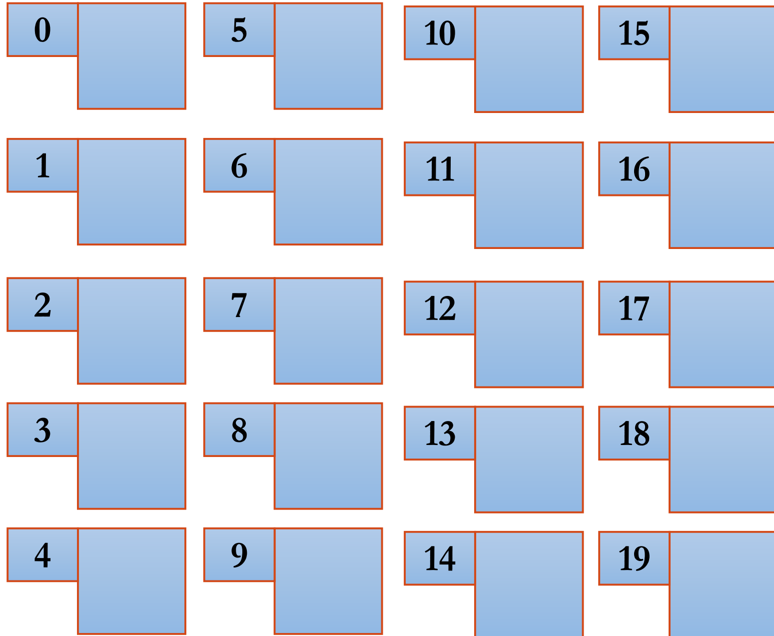


$g = 3$

e.g., L
insertions

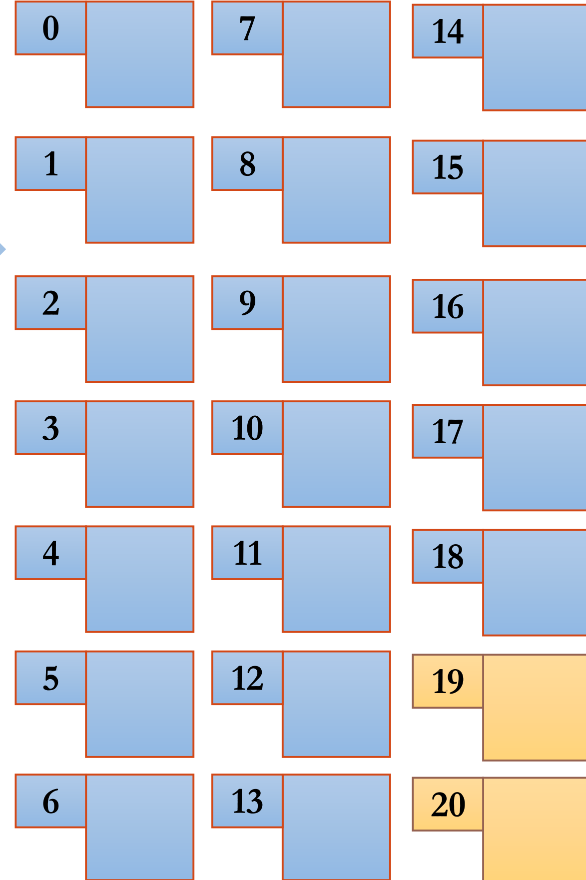


LHPE-RL – EXAMPLE



reorganization

The reorganization is only **virtual** to form the new groups of pages records of which will be redistributed together. No records are moved physically at this step.



LHPE-RL – EXAMPLE

- b = 3
- $h_0(k)$ = k mod 4
- $h_1(k)$ = k mod 3
- $h_2(k)$ = (k div 3) mod 3
- …

- Insert: 17, 9, 43, 21, 49, 35, 70, 52, || 40, 13, || 5, 8, || 37
  - General strategy: splitting || after 2 inserts
  - First delayed split after 8th insert (enough space)

A | 0 | | 2 |
B | 1 | | 3 |

Insert 8 values using $h_0(k)$

A | 0 | 52 | 2 | 70
B | 1 | 17 9 21 | 3 | 43 35
49

Split group A, use $h_1(k)$ for it

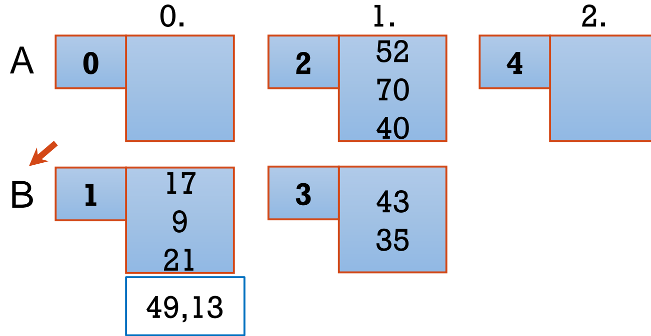
0. 1. 2.

A | 0 | | 2 | 52 70 | 4 |
B | 1 | 17 9 21 | 3 | 43 35
49

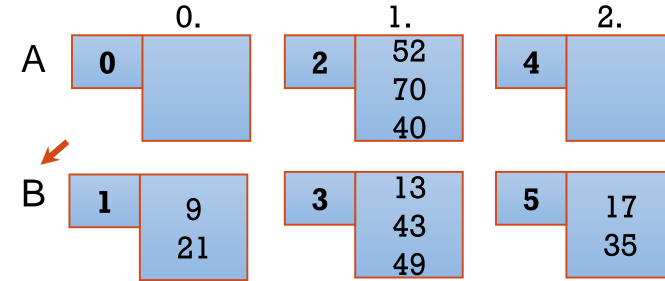
LHPE-RL – EXAMPLE

$h_0(\mathrm{k}) = \mathrm{k} \bmod 4$
$h_1(\mathrm{k}) = \mathrm{k} \bmod 3$
$h_2(\mathrm{k}) = (\mathrm{k} \operatorname{div} 3) \bmod 3$

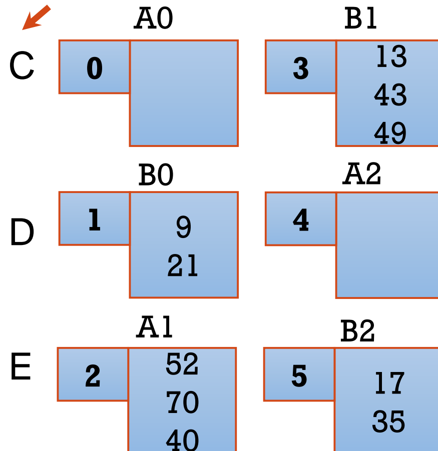
Insert 40, 13. Use $h_1(\mathrm{k})$ for the 1. group



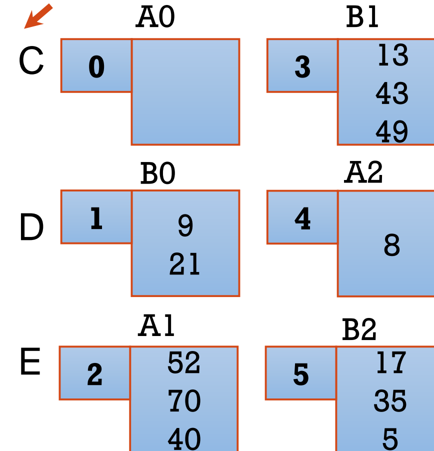
Split group B, use $h_1(\mathrm{k})$ for it



Redistribute the groups



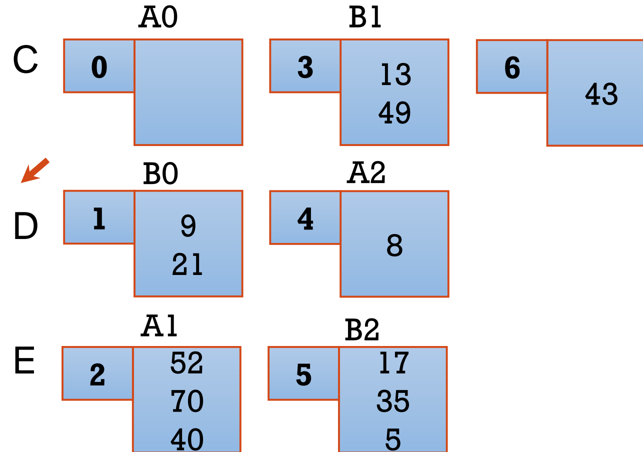
Insert 5, 8. Use $h_1(\mathrm{k})$ and ORIGINAL indices of the groups



LHPE-RL – EXAMPLE

$h_0(k)$ = k mod 4
$h_1(k)$ = k mod 3
$h_2(k)$ = (k div 3) mod 3

Split group C, use $h_2(k)$ for it



LHPE-RL – ADDRESSING

- The file undergoes series of redistributions and splits and so do the records in the pages
- The method assumes one initial hashing function $\boldsymbol{h_0}$ and series of independent hashing functions $\boldsymbol{h_i{:}K{\rightarrow}\{0\dots g\}}$ being used when splitting to identify the offset of records in each group
- To identify a position of a record in the file the sequence of splits and redistributions has to be "replayed"

LHPE-RL – ADDRESSING

1. At stage 1, a record with key $\boldsymbol{k}$ is inserted into a page determined by initial hashing function $\boldsymbol{h_0}$ $(\boldsymbol{k})$
2. When the split pointer gets to the group where $\boldsymbol{k}$ resides, the records are redistributed using hash function $\boldsymbol{h_1}$ $(\boldsymbol{k})$ (mapping to the space <0;$g-1$> and thus the record moves into page $\boldsymbol{p_1}$=$\boldsymbol{h_0}$ $(\boldsymbol{k})$ % $\boldsymbol{s_1}$+$\boldsymbol{h_1}$ $(\boldsymbol{k})*\boldsymbol{s_1}$
3. After the redistribution (stage = 2), page $\boldsymbol{p_1}$ gets into group $\boldsymbol{g_2}$=$\boldsymbol{p_1}$ % $\boldsymbol{s_2}$
4. When the split pointer reaches $\boldsymbol{g_2}$, $\boldsymbol{h_2}$ is used to get the new addresses for records in pages in $\boldsymbol{g_2}$ (and therefore also page p_1 where the record with key k resides) → $\boldsymbol{p_2}$=$\boldsymbol{g_2}$+$\boldsymbol{h_2}$ $(\boldsymbol{k})*\boldsymbol{s_2}$
5. The process iterates until the last stage $\boldsymbol{d_L}$ is reached
6. If $\boldsymbol{g_L}$ is greater or equal than the split pointer position the desired page is $\boldsymbol{p_{L-1}}$, otherwise we need moreover to compute $\boldsymbol{p_L}$ using h_L