PRINCIPLES OF DATA ORGANISATION

Cormack

MOTIVATION

- Key/pointer pairs ~ index
- Hashed file organisation
- **External static** hashing
  - We know how much data we can expect

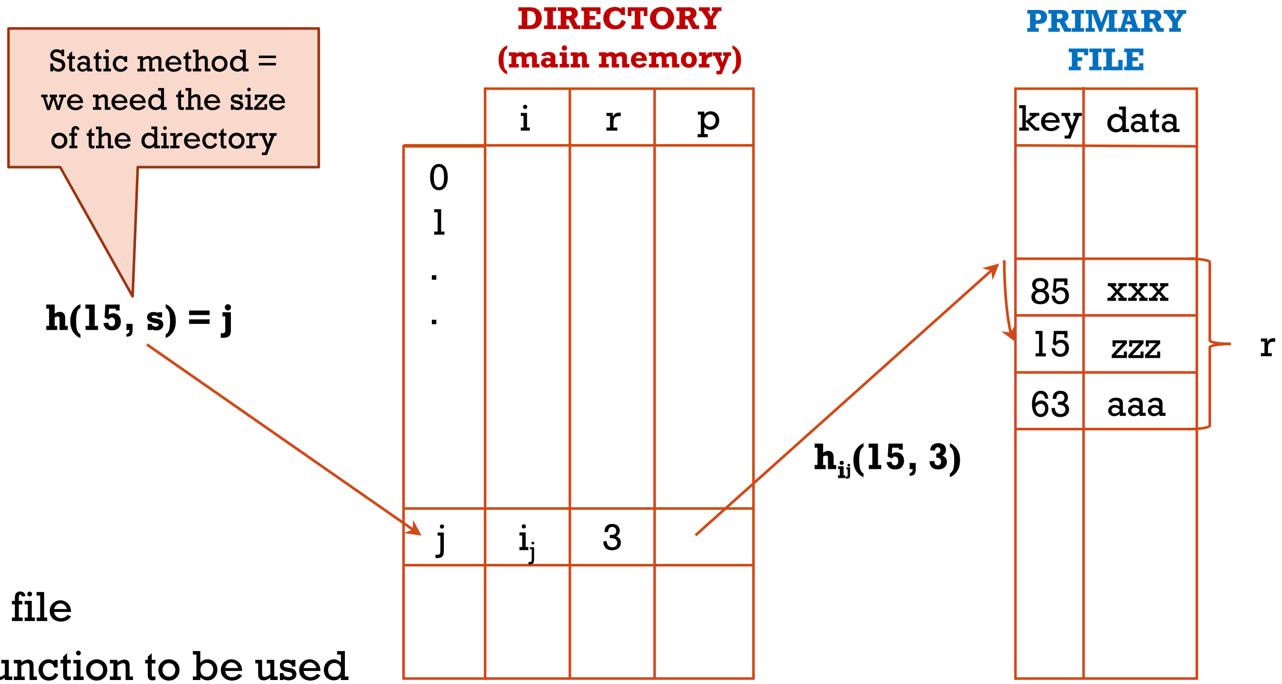
CORMACK

- Perfect hashing ~ no overflow policy required
- Requires additional ***O(N)*** space (directory)
  - A level of indirection
- Requires a set of hash functions
  - One function is used as the initial hash function ***h(k,s)*** to access the directory
    - ***s*** – the size of the directory
  - Another function is used to hash collided records into a continuous space
    - We have a set of such functions $\mathbf{h_i(k, r)}$
      - ***i*** – index of used hashed function
      - ***r*** – number of referenced records in the hash table
- Idea: first, we split the hashing problem into smaller ones, and then we solve a "smaller" hashing problem (i.e., finding a perfect hashing function) locally for a small number of values
- Presumes a low number of collisions → it is possible to find a perfect hashing function for the collided records in a reasonable time

CORMACK – ALGORITHM

```
typedef struct {
  int p, i, r;
} dLine;

typedef struct {
  KEY_TYPE key;
  DATA data;
} pfLine;
```



s – size of the directory

p – pointer to the primary file

i – index of perfect hash function to be used

r – number of colliding records in the primary file

CORMACK – EXAMPLE

- $h(k) = k \bmod 7$
- $h_i(k, r) = (k >> i) \bmod r$

$h(14) = 0$
Find ***i*** s.t.
$h_i(14, r) = (14 >> i) \bmod r = 0$
$r = 1$ (number of colliding records)

$h(17) = 3$
Find ***i*** s.t.
$h_i(17, r) = (17 >> i) \bmod r = 0$

	i	r	p
0	0	1	0
1			
2			
3	0	1	1
4			
5			
6			

0	14
1	17
2	
3	
4	
5	
6	
7	
…	

CORMACK – EXAMPLE

- $h(k) = k \bmod 7$
- $h_i(k, r) = (k >> i) \bmod r$

$h(10) = 3$
Find ***i*** s.t.
$h_i(10, r) <> h_i(17, r)$
$r = 2$

try $i = 0$:
$h_0(10, 2) = 0$
$h_0(17, 2) = 1$

	i	r	p
0	0	1	0
1			
2			
3	0	~~1~~ 2	~~1~~ 2
4			
5			
6			

0	14
1	~~17~~
2	10
3	17
4	
5	
6	
7	
…	

> Find new space (or extend the current)

Try to insert 21 and 28

CORMACK – EXAMPLE

- $h(k) = k \bmod 7$
- $h_i(k, r) = (k >> i) \bmod r$

	i	r	p
0	0	3	4
1			
2			
3	0	2	2
4			
5			
6			

0	~~14~~
1	~~17~~
2	10
3	17
4	21
5	28
6	14
7	
…	

$h(42) = 0$

Find ***i*** s.t.

$h_i (21, r) <> h_i (28, r) <>$

$h_i (14, r) <> h_i (42, r)$

$r = 4$

try i = 0:

$h_0 (14, 4) = 2$

$h_0 (21, 4) = 1$

$h_0 (28, 4) = 0$

$h_0 (42, 4) = 2$

try i = 1:

$h_1 (14, 4) = 3$

$h_1 (21, 4) = 2$

$h_1 (28, 4) = 3$

$h_1 (42, 4) = 1$

… try i = 2, 3, 4, 5

If it does not help, increase r (i.e., increase space = use more slots) … try r = 5 and i = 0

CORMACK – ALGORITHM

```
bool INSERT(dLine dir[], pfLine pf[], pfLine rec)
{
 int s = dir::size();
 int j = h(rec.key,s);
 if (dir[j].r == 0)  {
  int posNew = FREE(pf, 1);
  pf[posNew] = rec;
  dir[j].p = y;
  dir[j].i = -1;
  dir[j].r = 1;
 }  else   {
  int r = dir[j].r,
  int p = dir[j].p;
  if (CONTAINS(pf, p, r, rec.key))
   return FALSE;
```

```
/* find a hash function with index m, such that
h_m(rec.key, r+1) != h_m(pf[p].key, r+1) !=
h_m(pf[p+1].key, r+1) != … != h_m(pf[p+r-1].key,
r+1) */
  int m = FIND_HASH_F(pf, p, r, rec.key);

// allocate space for colliding records and the
new record
  int posNew = FREE(r+1);

// copy the existing colliding records into new
space
  for (int i = 0; i < r; i++)   {
   pf[posNew+h_m(pf[p+i].key, r+1)] = pf[p+i];
   ERASE(pf, p+i); // free memory
  }
  dir[j].p = newPos;
  dir[j].i = m;
  dir[j].r = r+1;
 }
}
```

CORMACK – ALGORITHM

```
void ACCESS(dLine dir[], pfLine pf[], KEY_TYPE k, int &pfPos, bool &found) {
  int s = dir::size();
  int j = h(k,s);
  if (dir[j].r == 0)
    found = FALSE;
  else {
    int ij = dir[j].i;
    pfPos = dir[j].p + h_ij(k, dir[j].r);
    if (pf[pfPos].key != k)
      found = FALSE;
    else
      found = TRUE;
  }
}
```

CORMACK – ALGORITHM

- The choice of independent hashing functions:

$$h_i\,(k,r)=(k\ mod\ (2i+100r+1)),\ where\ k \gg 2i+100r+1$$

- for a given **i** and a set of ***r*** keys $\boldsymbol{h_i}$ is a perfect hash function
- **r** needs to be small enough to be able to find a suitable ***i*** in a reasonable time
- If there are too many collisions, we can use ***r*^2** instead