TO: MSPM Distribution
FROM: Carole A. Cushing
SUBJ: BG.7.00
DATE: 08/04/67

Minor corrections have been made to MSPM section BG.7.00 to make it agree with the current declarations being used for directories.

Published: 08/04/67
(Supersedes: BG.7.00, 02/03/67
BG.7.00, 05/11/66)

Identification

Directory Data Base
C. A. Cushing

Purpose

The directory data bases are special segments maintained by the directory control module which contain information about every segment on secondary storage. Each directory contains a list of entries and each entry consists of a description of a particular segment or of another directory entry. Those entries which contain information about segments are called branches and those entries which contain information about other directory entries are called links. Each segment has only one branch in one directory associated with it. Each directory entry may have any number of links in any number of directories associated with it.

Since a directory is a common data base, certain interlocks and switches have been placed in the directories. By observing common rules about the interlocks the various primitives of Directory Control are able to guarantee the integrity of the data with which they deal. Section BG.15 describes the mechanism for locking data bases and for blocking those processes which are unable to lock a data base already locked by another process.

Contents of a Directory

The following is an outline of the items of information kept for each entry in a directory and the common information kept for the whole directory.

Data Base Outline

I. general directory information

    A. directory lock

        1. interlock flag

I. A. 2. no-more-readers switch
3. read-count
B. hash table
1. number of entries used in table
2. total length of table
3. table entries
C. slot tables
D. slot number of latest vacated
1. branch
2. link

II. common access control list (CACL)
A. interlock flag
B. vacant switch
C. date and times
1. last used
2. last modified
3. last dumped
D. number of entries
E. CACL entries
1. user name
2. mode
3. protection list and gate list
4. trap procedure and argument list

III. branch information

A. fixed information

1. vacant-entry switch
2. main interlock flag
3. entry-type
   a. directory
   b. non-directory
4. unique identification
5. date and times
   a. branch last modified
   b. segment last dumped
6. option switches
7. usage status
   a. not-used
   b. being read
   c. being written
   d. being data-shared
8. no-more-users switch
9. usage count
10. length of segment in bits
11. maximum length of segment
12. retention date
13. consistency flag

B. active information

1. interlock flag
2. segment active switch
3. activity indicator and date last changed

III. B. 4. current length of segment

5. move identification

6. date and times

a. segment last used

b. segment last modified

7. high limit

8. low limit

9. account number

10. device identification

11. file map size

C. variable length information

1. list of names for the branch

2. retrieval trap procedure, argument list and switch

3. system trap procedure, argument list and switch

4. access control list (ACL)

a. ACL entries

(1) access-control name

(2) mode

(3) protection list and gate list

(4) trap procedure name and argument list

5. file map

IV. link information

A. fixed information

1. vacant entry switch

2. interlock flag

IV. A. 3. unique identification

4. date and times

a. link last used

b. link last modified

c. link last dumped

B. variable length information

1. list of names for the link

2. path name of entry to which this is a link

Explanation of Outline

I. general directory information

directory lock

Three flags are used to implement process interlocking for the directory data base. (See BG.18 for a complete description of process locking and blocking). The interlock is used to flag whether the directory is or is not locked for modification. If the directory is locked, interlock contains the identification number of the process on whose behalf it is locked. Otherwise, it is zero. The read-count is used to keep the count of the total number of processes currently in the directory for reading purposes only. If a process wished to modify a directory but cannot because there are processes reading it, this process can set the no-more-readers switch ON and then go blocked waiting for the current readers to leave (read-count=0). While the no-more-readers switch is ON, no further processes are allowed to read the directory.

Waking processes which are blocked waiting for the interlock flag or read-count to become zero is explained in BG.18.

hash table

The hash table is used to find an entry in a directory given its symbolic name. There is a location in the hash table for each name of every entry in the directory.

For each name, its associated location in the hash table contains a slot number, i.e., an index into a slot table (explained below) in the directory, identifying the location in the slot table which contains the pointer to the entry with that name. A count is kept of the number of locations in the hash table that are currently or have been used to point to an entry.

Each hash table location contains a signed number where the magnitude of the number denotes a location in a slot table. If the sign is +, the slot points to a branch through a pointer in the branch slot table. If -, to a link through a pointer in the link slot table.

slot tables

There are two slot tables in a directory, a link slot table and a branch slot table. These tables contain pointers to each branch and link in the directory. The first branch (link) to be created in a directory will have its pointer stored in the first slot in the branch (link) slot table. As long as that branch (link) remains in the directory its pointer will remain in the first slot. As successive branches (links) get created in the directory successive slots will be filled with pointers to them. As branches (links) get freed, their slots will be zeroed. These zeroed slots now become candidates to be used for newly created branches (links). A branch or link in the heirarchy can now be located by either a path name, a string of symbolic names of each entry from the root to the branch, or a slot name (positional notation), a string of slot numbers of each entry from the root to the branch.

slot number of latest vacated entry

Each number is a pointer to the bottom of a linked list of vacant directory entries. One, if non-zero, is the slot number of the most recently vacated branch. This branch in turn contains the slot number of the next most recently vacated branch, etc. If this initial index is zero, there are no vacant branches.

The other, if non-zero, is the slot number of the most recently vacated link. This link in turn contains the slot number of the next most recently vacated link, etc. If this initial index is zero, there are no vacant links.

II. common access control list (CACL)

The common access control list contains access information common to all entries in the directory.

vacant switch: If this switch is ON, there is no CACL for this directory

date and times: There are three dates recorded in the CACL in units of microseconds since the year 1900. The date/time when the CACL was last used, last modified, and last dumped

CACL entry

user name: This is a symbolic user identification consisting of a personal name, project identification and login identification

mode: The mode is a five bit flag indicating trap, read, execute, write, and append attributes

protection list: This is a list of numbers used to determine the access bracket, call bracket and gates of all segments in the directory for this user (brackets and gates are defined in section BD.9, Protection of the Supervisor, and the structure of this list is defined in section BG.9, Access Control)

trap procedure: This is the path name of a procedure to be called if the trap attribute is ON

argument list: The arguments are in a character string with appropriate separators (those recognized by the Shell) between arguments and are passed through the Access Control Module to the trap procedure

III. branch information

Branches are directory entries which point directly to segments.

III. fixed information:

This information may be read or written only when the branch is locked to the process, except for the interlock flag itself and the unique identifier which may be read whether the branch is locked or not.

vacant-entry switch: If this switch is ON, the following information is meaningless, except for the usage count which is interpreted as the slot number of the next vacant branch. If OFF, meaningful information is stored in this entry.

main interlock flag: If this is non-zero, then the branch is locked by the process whose process id. is its value.

directory switch: If this switch is ON, the branch points to a directory segment. If OFF, the branch points to a non-directory segment.

unique identification: This is a unique number within all versions of Multics which defines the precise copy of the segment pointed to by this branch.

date and time: The date/time in number of micro-seconds since the year 1900 when the branch was last modified and the date/time segment to which the branch points was last dumped.

options switches: This is a string of two switches indicating the setting of the two options, copy and relate, for the segment to which this branch points. Both of these options are interpreted by the Segment Housekeeping Module.

usage status: This indicates the current state of the file, not-used, used for reading, used for writing, or used for data-sharing.

III. no-more-users-switch: If this switch is ON, no more processes will be allowed to use this segment. This switch is set ON by a process going blocked to wait for the usage count to become zero. When the usage count becomes zero, this switch is turned OFF and blocked processes are awakened.

usage count: This is a count of the number of processes which are using the segment for the current status. If that status is read or data-share, or the process id of the process which is using the segment for writing

bit count: This is a count of the number of bits of information in the segment (EOF mark).

maximum length: This is a preset maximum of the segment expressed in units of 1024 words.

retention date: This is the date after which the branch and segment are to be deleted.

consistency flag: This flag specifies to the backup system that the user does or doesn't want the subtree beneath this branch to be dumped consistently (see BH.2.00). It tells the user that the subtree is currently consistent, is waiting to be dumped in a consistent state, or is inconsistent, i.e., dump aborted while in subtree or entire subtree was not reloaded.

active information

This information may be read if the main branch lock and the active lock are set for this process and written if the active lock is set for the process.

lock: If this is non-zero, the active information is readable and writeable by the process whose process id. is its value. If zero, the active information may not be read or written.

segment active switch: If this switch is ON, the segment is active and the following information must be updated since it may not be accurate. If OFF, the segment is inactive and the following information is accurate.

activity indication: This is a measure of i/o activity due to paging and is set and interpreted by multilevel (BH.1.01).

III. current length: This is the current length of the segment in units of 64 words (users will see this in units of 1024 only).

move identification: This identifies the device to which this segment is being moved, e.g., drum, disk, etc.

high and low limits: These are two preset constants by which the user specifies to the multilevel system the range of devices on which he wishes this segment to reside.

account number: This is the account number to which storage for this segment is charged

device identification: This identifies the device on which this segment resides, e.g., drum, disk, etc.

file map size: This is the size of the file map (see below) for the segment.

variable information

The following information involves a variable amount of space in the directory. In order to read the information the main branch must be locked by the process, and in order to modify it the directory must be locked also (see the discussion below).

list of names: The list contains the names by which the branch is known.

retrieval trap and switch: This trap consists of the path name and argument list for a procedure to be called when the segment is referenced by a user if the switch is ON. The switch is set ON after the segment has been removed to off-line storage by the backup or multilevel systems.

system trap and switch: This trap consists of the path name and argument list for a procedure which is called (if the switch is ON) each time this branch is referenced by a user.

file map: The file map is used by the DIM to determine the physical location of the segment on an offline device.

III. link information

Links are directory entries which point to other directory entries (those items in a link which are identical to items in a branch are defined the same).

fixed information

date and time: These are three date/times indicating when this link entry was last used, last modified, and last dumped.

variable information

path name: This is the symbolic path name of the entry to which this link points

PL/I Implementation of Directory Data Bases

Directories are data bases common to many processes. They consist of sets of information where the size of some sets is of fixed length and the size of others is variable length. These sets of variable size (eg., the set of names for a branch in a

directory) contain information which when changed may increase, decrease, or not affect the size of the sets. In order to handle information of this type, there exist routines which will allocate and free storage for the sets. (NOTE: Because the pool of free storage in the directory may be changed when this type of information is being modified, the directory must be interlocked by the modifier.) When a set of certain size has been allocated, a pointer (ITS pair) to the base storage location for this set is returned. This pointer is used to fill in and refer to this set in its allocated area. In order to remember where this set of information is located, there is a need to save the pointer to it in the directory. Since a pointer contains a segment number and a directory is a common data base, pointers (ITS pair) cannot be stored in directories.

Two routines exist which will handle this problem. One routine called rel will create a relative pointer (offset without segment number) from a pointer. This relative pointer may then be stored in a directory.

The other routine called ptr will create a pointer (ITS pair) from a relative pointer and the base pointer of the directory in which the relative pointer was found (see BY.14).

Figure 1 is a diagram of the various sets of information in a directory giving a "clue" to how they are allocated and accessed.

The variable dp is a pointer variable (ITS pair) in automatic storage and points to the base of a directory segment. The layout of the entire segment is defined by the following statement:

```
dcl 1 dir based(dp),

    2 ilock(3)bit(36),          /* lock, nomore, readcount */

    2 uid bit(70),              /* unique identifier of directory */

    2 (tbcount,tlcount)fixed bin(17),
                                /* size of branch(link)slot table */

    2 (tbdate,tldate,tcdate)bit(52),
                                /* date and time above totals were
                                   last updated and CACL was last
                                   modified */
```

```
    2 (bcount,lcount)fixed bin(17),
                              /* number of used branches
                                 (links) */

    2 (vbcount,vlcount)fixed bin(17),
                              /* number of vacant branches
                                 (links) */

    2 hrtp bit(18),           /* rel pointer to hash table */

    2 htsize fixed bin(17),   /* size of hash table */

    2 htused fixed bin(17),   /* number of used entries in the
                                 hash table */

    2 (vsbn,vlsn)fixed bin(17),
                              /* vacant branch(link)slot
                                 number */

    2 (bsrp,lsrp)bit(18),     /* rel pointer to branch(link)slot
                                 table */

    2 caclrp bit(18),         /* rel pointer to CACL */

    2 var area ((1));
```

The following are structure declarations for the various sets of information to be found in a directory. These sets will be allocated in dir.var.

See section BG.1.00 for the distinction between declaration for reference and declaration for allocation. All following structures which are starred refer to this reference.

The pointer variable caclp points to the CACL of the directory.

```
    caclp = ptr (dp, dp->dir.caclrp);

    dcl 1 cacl based(caclp),

        2 ilock bit(36),

        2 clrp bit(18),       /* rel ptr to first CACL entry */

        2 vacant bit(1),

        2 (dtu,dtm,dtd) bit(52);
```

The relative pointer, clrp, points to the first CACL entry in a linked list of entries. These entries have the following declaration:

```
dcl 1 clentry based (clp),          /* clp=ptr(dp,
                                       caclp->cacl.clrp) */

    2 userid,

        3 project_id char (24),

        3 name char (24),

        3 login_id char (2),

    2 mode bit (5),

    2 plistrp bit (18),             /* rel pointer to
                                       protection list */

    2 gatesrp bit(18),              /* rel pointer to list
                                       of names of gates */

    2 traprp bit (18),              /* rel pointer to trap
                                       procedure and argument
                                       list */

    2 clrp bit (18);                /* rel pointer to next
                                       control list entry */


*dcl 1 protect based(plistptr),     /* plisptr=ptr(dp,
                                       clp->clentry.plistrp */

     2 pad bit(1),

     2 listsize bit(17),

     2 list(plistptr->protect.listsize)bit (18);

*dcl 1 trapproc based(tp),          /* tp=ptr(dp,clp->clentry
                                       .traprp */

     2 size fixed bin(17),

     2 string char (tp->trapproc.size
```

```
*dcl 1 gates based (gp),          /* gp=ptr(dp,clp->clentry
                                     .gatesrp) */

     2 size fixed bin(17),

     2 nnrp bit(18),              /* rel ptr to next
                                     gate name */

     2 name char(gp->gates.size);
```

The hash table in a directory contains an array of slot numbers. The slot numbers refer to relative locations within the branch slot table (if type of slot number is 0) or the link slot table (if the type of the slot number is 1).

The hash table is located by the pointer htp where
htp = ptr (dp, dp->dir.htrp);

```
dcl 1 hashtbl (dp->dir.htsize) based (htp),

    2 vacated bit (1),            /* =1 if location had
                                     been used */

    2 type bit (1),               /* =1 if link, =0 if
                                     branch */

    2 slotno bit (17);            /* =0 if location now
                                     empty */
```

The branch and link slot tables have the same structure.

The pointer sp, where sp = ptr (dp, dp->dir.bsrp);

or sp = ptr (dp, dp->dir.lsrp);

points to the branch or link slot table.

```
*dcl  1 slots based (sp),

      2 pad bit (1),

      2 size bit (17),            /* =dp->dir.tbcount or =
                                     dp->dir.tlcount */

      2 rp(sp->slots.size)bit(18); /* array of relative poin-
                                     ters to branches or
                                     links */
```

The following code might be used to get the pointer to the entry with a name which hashes into the third location in the hash table:

```
htp=ptr(dp,dp->dir.htrp);

i=htp->hashtbl(3).slotno;          /*i=slot number in third
                                   location of hash table*/

if htp->hashtbl(3).type            /*type of entry pointed
then                               to by this slot number */
sp=ptr(dp,dp->dir.bsrp);           /* is a branch */

else
sp=ptr(dp,dp->dir.lsrp);           /* is a link */

ep=ptr(dp,sp->slots.rp(i));        /*ep=ptr to ith branch or
                                   link */
```

The branches of a directory are pointed to by the pointer ep where:

```
ep=ptr(dp,sp->slots.rp(i));        /*sp=ptr(dp,dp->dir.bsrp)
                                   */

dcl 1 branch based(bp),

    2 ilock bit(36),               /*branch lock */

    2 activinfo bit(36),           /*lock on active info in
                                   branch */

    2 uid bit(70),

    2 vacant bit(1),               /*if=1,this branch is
                                   vacant */

    2 dirsw bit(1),                /*if=1,branch points to a
                                   directory */

    2 usage bit(2),                /*usage status */

    2 usagect bit(17),

    2 nomore bit(1),

    2 (dtd,dtbm) bit(52),          /* date and time seg
                                   dumped, branch modified
                                   */
```

```
        2 rd bit(52),                   /*retention date */
        2 options bit(2),               /*two options switches,
                                          copy and relate */
        2 consistsw bit(2),             /*consistency flag */
        2 ml bit(9),                    /*max length of seg-
                                          ment */
        2 bc bit(24),                   /*no. of bits of informa-
                                          tion in segment */
        2 pad1 bit(3),                  /*used to prevent items
                                          from straddling word
                                          boundary */
        2 activsw bit(1),               /*=1, if segment active*/
        2 actind bit(17),               /*i/o activity indicator
                                          */
        2 actime bit(52),               /*time above indicator
                                          updated */
        2 pad2 bit(2),
        2 (dtu,dtm) bit(52),            /*date and time segment
                                          used and modified */
        2 move bit(4),                  /*identification of dev-
                                          ice segment being moved
                                          to */
        2 acct bit(36),                 /*account number */
        2 cl bit(13),                   /*current length of seg-
                                          ment */
        2 did bit(4),                   /*device id. */
        2 pad3 bit(2),
        2 llim bit(17),                 /*multi-level limit */
        2 pad4 bit(1),
        2 hlim bit(17),
```

```
        2 pad5 bit(1),
        2 fmsize bit(18),           /*size of file map
                                    pointer */
        2 systrp bit(18),           /*rel pointer to system
                                    trap */
        2 systsw bit(1),            /*=1,if trap to be
                                    executed */
        2 nnames bit(17),           /*no. of branch names */
        2 bnrp bit(18),             /*rel pointer to first
                                    name of branch */
        2 aclrp bit(18),            /*rel pointer to first
                                    acl entry ("clentry"
                                    type) */
        2 fmrp bit(18),             /*rel ptr to file map */
        2 retrieve bit(18),         /*rel ptr to retrieval
                                    trap */
        2 retrievesw bit(1);        /*=1,if trap to be
                                    executed */
```

The first name in the linked list of names for a branch is pointed to be np where

```
        np=ptr(dp,bp  branch.bnrp);
    *dcl 1 names based(np),
        2 pnrp bit(18),             /*rel ptr to previous
                                    name */
        2 nnrp bit(18),             /*rel ptr to next name */
        2 size bit(17),
        2 name char(np->names.size);
```

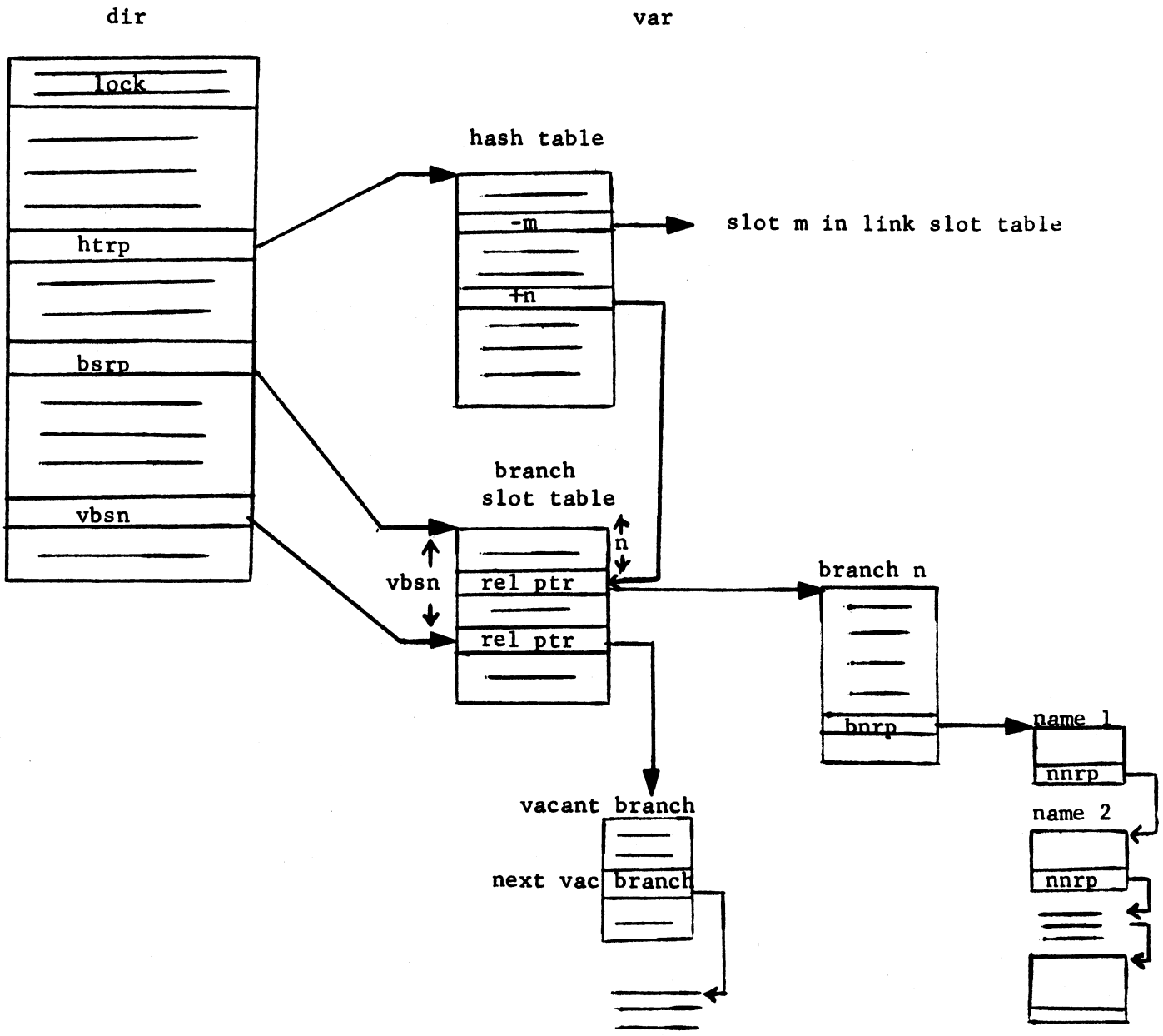
The links of a directory are pointed to by the pointer ep.

```
    ep=ptr(dp,sp->slots.rp(i));     /* ,p=ptr(dp->dir.lsrp)*/
```

```
dcl 1 link based(ep),
    2 ilock bit(36),
    2 vacant bit(1),
    2 pad1 bit(1),
    2 uid bit(70),
    2 pad2 bit(1),
    2 nnames bit(17),
    2 lnrp bit(18),          /*relative pointer to
                               link names */
    2 pad3 bit(1),
    2 pnsize bit(17),        /*total char count of
                               path name */
    2 pnrp bit(18),          /*relative pointer to
                               path name */
    2 pad4 bit(4),
    2 (dtu,dtm,dtd) bit(52);
```

To get the path name of the entry to which a link points set pnp.

```
pnp = ptr (dp, ep->link.pnrp);
dcl pathname char (ep->link.pnsize) based (pnp);
```


dir
var
lock
htrp
bsrp
vbsn
hash table
-m
+n
slot m in link slot table
branch
slot table
n
vbsn
rel ptr
rel ptr
branch n
bnrp
name 1
nnrp
name 2
nnrp
vacant branch
next vac branch