```
/*
Print a list of pages to read to answer given MAH queries.

Command line args give file parameters:
* n = number of attributes in tuples
* d = file depth (# bits used in hash)
* cv = choice vector (e.g. 1,2,1,2,3,...)

The main loop reads queries one per line from stdin.
Each query is a list of space-separated items.
Each item is either a value or a "?", where "?" means 'unknown'.

On choice vectors (CV):
Each number in the CV indicates an attribute (1..n).
There must be at least d+1 numbers in the choice vector.
Bits from each attribute are added to the CV from right-to-left.

Example:

n=4, d=5, cv=1,1,2,3,1,2,3

The choice vector is as follows:
* hash bit 0 comes from bit 0 of attribute 1
* hash bit 1 comes from bit 1 of attribute 1
* hash bit 2 comes from bit 0 of attribute 2
* hash bit 3 comes from bit 0 of attribute 3
* hash bit 4 comes from bit 2 of attribute 1
* hash bit 5 comes from bit 1 of attribute 2
* hash bit 6 comes from bit 1 of attribute 3

Note: attribute 4 is not involved in the hash value at all.


Example queries (for relation R(a,b,c,d)):

?,?,?,?    represents  select * from R
1,?,abc,?  represents  select * from R where a=1 and c='abc'
?,5,xyz,7  represents  select * from R where b=5 and c='xyz' and d=7

*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "bits.h"
#include "hash.h"

// Global constants

#define MAX_ATTRS 8
#define MAX_DEPTH 30
```

```
#define MAX_CVLEN (MAX_DEPTH+2)

// File parameters ... done as globals (tsk, tsk)

typedef struct { int attr; int bit; } CVitem;
typedef struct { int known; uint32 hash; } Qitem;

int    n;               // number of attributes;
int    d;               // file depth (use d or d+1 hash bits)
CVitem cv[MAX_CVLEN];   // array of choice vector items
int    ncv;             // number of items in choice vector
Qitem  q[MAX_ATTRS+1];  // parsed version of query

// Functions (forward referenced)

void usage(char *);
int  makeChoiceVector(char *); // sets "cv", "ncv"
int  parseQuery(char *);       // reads "n", sets "q"

// ===== Main =====

int main(int argc, char **argv)
{
        char line[200];  // input line buffer

        // process command-lines args and set things up

        if (argc < 4)
                usage("Not enough parameters");
        if (sscanf(argv[1], "%d", &n) != 1 || n > MAX_ATTRS)
                usage("Invalid number of attributes");
        if (sscanf(argv[2], "%d", &d) != 1 || d > MAX_DEPTH)
                usage("Invalid file depth");
        if (!makeChoiceVector(argv[3]))
                usage("Invalid choice vector");

        // read queries and display required pages

        uint32 known = 0, unknown = 0;


        printf("query> ");
        while (fgets(line,200,stdin) != NULL) {
                if (strcmp(line,"quit\n") == 0)
                        break;
                if (!parseQuery(line)) {
                        fprintf(stderr, "Invalid query\n");
                }
                else {
                        // Hint: what your code should do ...
                        // determine unknown bits from query and
choice vector
```

```
                        // set known bits from query hashes and
choice vector
                        // for each configuration of unknown bits
                                // generate new page number

            known = 0;
            unknown = 0;
                        // XXX your code goes here XXX
                        for (int i = 0; i < ncv; i++) {
                            if (q[cv[i].attr].known == 1) {
                                if (bitIsSet(q[cv[i].attr].hash,
cv[i].bit))
                                    known = setBit(known, i);
                            } else {
                                unknown = setBit(unknown, i);
                            }
                        }
//                      debugging ...
//                      char out[200];
//                      showBits(known,out); printf("Known:   %s\n",
out);
//                      showBits(unknown,out); printf("Unknown:
%s\n", out);
            printf("Pages:");

            int nstars = 0;
            for (int i = 0; i < ncv; i++) {
                if (bitIsSet(unknown,i)) nstars++;
            }

            // for all possible combinations of 2^nstars bits
            int counter;
            for (counter = 0; counter < (1<<nstars); counter++) {
                int i = 0, j = 0;
                uint32 b = known;
                //showBits(b,out); printf("Starting with: %s\n",out);
                for (i = 0; i < ncv; i++) {
                    //printf("checking for * at %d\n",i);
                    if (bitIsSet(unknown,i)) {
                        //printf("found * at %d\n",i);
                        // fit next bit from counter into hash
                        if (counter & (1<<j)) {
                            //printf("counter has 1 at %d\n",j);
                            b = setBit(b,i);
                        }
                        j++;
                    }
                }
                //showBits(b,out); printf("Bucket: %s\n",out);
                printf(" %d", b);
            }
```

```
            printf("\n");
                }
                printf("query> ");
        }
        printf("\n");
        return 0;
}


// ===== Supplied functions =====

// makeChoiceVector()
// reads a string giving choice vector data
// parses string and sets values of "cv" and "ncv"

int makeChoiceVector(char *in)
{
        char *w;
        int attr;
        int attrbit[MAX_ATTRS] = {0,0,0,0,0,0,0,0};

        ncv = 0;
        for (w = strtok(in,","); w != NULL; w = strtok(NULL,",")) {
                if (sscanf(w,"%d",&attr) != 1)
                        return 0;
                if (attr < 1 || attr > n)
                                return 0;
                cv[ncv].attr = attr;
                cv[ncv].bit  = attrbit[attr]++;
                ncv++;
        }
        if (ncv < d) usage("Choice vector too short");
#if 1
        // debugging: comment out if you don't need it
        int i;
        printf("CV = < ");
        for (i = 0; i < ncv; i++)
                printf("(%d,%d) ", cv[i].attr, cv[i].bit);
        printf(">\n");
#endif
        return 1;
}

// parseQuery()
// reads a string giving query data
// parses string and set value of "q"
// checks that there are "n" items in string

int parseQuery(char *in)
{
        char *w;
        int attr;
```

```
        char buf[40]; // allows for 32 bits + spaces

        for (w = in; *w != '\n'; w++) /*skip*/;
        *w = '\0';
        attr = 1;
        for (w = strtok(in,","); w != NULL; w = strtok(NULL,",")) {
                if (w[0] == '?') {
                        q[attr].known = 0;
                        q[attr].hash  = 0;
                }
                else {
                        q[attr].known = 1;
                        q[attr].hash = hash_any((unsigned char *)w,
strlen(w));
                }
                attr++;
                if (attr > n+1) // too many values
                        return 0;
        }
        if (attr < n+1) // not enough values
                return 0;
#if 1
        // debugging: comment out if you don't need it
        int i;
        for (i = 1; i < attr; i++) {
                if (!q[i].known)
                        printf("q[%d] = ??\n",i);
                else
                        printf("q[%d] =
%s\n",i,showBits(q[i].hash,buf));
        }
#endif
        return 1;
}


void usage(char *message)
{
        if (message[0] != '\0')
                fprintf(stderr, "%s\n", message);
        fprintf(stderr, "Usage: ./pages  n  d  cv\n");
        exit(1);
}
```