Jonathan Buckner

COSC 3319.01 MWF

05/01/24

# Lab 3 Part A

Table of Contents:

Table of Results for Option C:

| | Old Hash Linear | | New Hash Linear | | Old Hash Random | | New Hash Random | |
|---|---|---|---|---|---|---|---|---|
| | 1st 25 | Last 25 | 1st 25 | Last 25 | 1st 25 | Last 25 | 1st 25 | Last 25 |
| **Min Probe** | 1 | 14 | 1 | 1 | 1 | 2 | 1 | 1 |
| **Max** | 16 | 96 | 2 | 20 | 3 | 15 | 2 | 15 |
| **Average** | 3.8 | 70 | 1.04 | 4.92 | 1.64 | 4.96 | 1.04 | 3.92 |
| **Theoretical** | .5 | 2.78571 | .5 | 2.78571 | 1.1126 | 1.94538 | 1.1126 | 1.94538 |

**Average number of probes to locate all keys in the table:**

| | Old Hash Linear | New Hash Linear | Old Hash Random | New Hash Random |
|---|---|---|---|---|
| Average all keys! | 35.15 | 2.3 | 7.22 | 1.94 |

## Discussion Questions:

A)The data for the keys can be found on page 2, and the contents of the entire hash table can be found on page 14.

B)Theoretical Probe count: **$E = ( 1 - a / 2 ) / ( 1 - a )$** where a = (number keys in the table) / (table size).
In this case, a = 100/128 or .78125, therefore its (1-.78125/2) / (1 - .78125) = 2.786 theoretical probes to find a random item in the hash table.

However, this number changes if want the success rate of the table at various stages. For instance, if you want the theoretical probe count at the first 25 probes, a = 25 / 128 or .1953. Then, its (1-.1953/2)/ (1 - .1953) = .5

C) In order to physically search for each individual key, I set up two functions, one to physically search for each key, adding to the count once it was found, and a function to utilize that search function for every slot in the hash table. The two functions can be found on page ___

Here is the output of the functions:
Actual Average Number of Probes to Find All Keys: 35.15.
This number is a lot higher than the projected theoretical average number of probes: 2.786. The reason is due to the poor distribution of the keys due to the sub-optimal hash function algorithm. This leads to lots of probes needed, especially due to the linear nature of the program. It has to loop all the way around the function in order to find an empty slot, as the slots are clustered around the beginning and end of the table, meaning it takes a long journey to find the concentrated empty slots that are in the center of the table.

D)

Calculating the theoretical expected number of probes used: **$E = - ( 1 / a ) \ln ( 1 - a )$** where a=(number keys in the table) / (table size).
In this case, a = 100/128 or .78125, therefore its E= -(1/.78125) ln(1-.78125) therefore E = 1.94538.
For the first 25 keys, a = 25/128 or .1953, yielding E = -(1/.1953) ln(1-.78125) or 1.1126

Utilizing the two functions yielded the actual number of probes as 7.22. The reason why the actual number of probes is higher than the theoretical projection of 1.94538 is for similar reasons to the linear probes, the poor quality of the hash function leads to a lot of clustering. However, there is a noticeable decrease in the difference between the theortical and empirical number of probes. The linear function had a difference of 32.364 whereas the random function has a difference of 5.27462. The reason the random function was more successful was because there was less primary clustering around the beginning and end of the function. There was a lot more secondary clustering scattered throughout the table, however, generally speaking the data was much more uniformly distributed than the linear function.

E)
The new hash linear function is over 15x more efficient than the old hash linear (from an average of 35.15 probes to only 2.3) and the new hash random function is a little over 3x more efficient (from 7.22 to 1.94). The reason why the new hash linear function is so much more efficient than the old one is that it isn't clustering around the beginning and end of the hash table, but the newer function isn't clustering at all, so whenever there is a collision it the probe doesn't have to go far, whereas sometimes in the old function the probe would have to go from the end of the hash table, then loop around, then into the center where the empty slots were. The results table can be found on page 1.
The old hash random function didn't have as much primary clustering, however, secondary clusters meant that it still wasn't very efficient. This meant that the new hash function still was more efficient than the old one.

In order to physically verify the hash function is working as intended, I will demonstrate by manually running through the algorithm for the first key and last key of the random hash function.

Given "ABCDEFGHIJKLMNOP"->
Here are the ASCII Codes:
Str[0] = A = 65
Str[1] = B = 66
Str[10] = K = 75
Str[3] and str[4] = D and E or 68 * 10 + 69 = 749
Str[6] and str[7] = G and H or 71 * 10 + 72 = 782
DE + GH =1531
1531 / 381 + A (which is 65) = ~4.01680 + 65 = 69.01680
69.01680 / 587 = .11764.
Finally:
Temp = B(66) + .11764 - K(75) = -8.88236. You then take the absolute value of that number, then mod it by 128. When you mod the value, the value is truncated and the floating points are lost, leaving us with 8. Which is where the string is placed in the algorithm.

Here is the last key:
Given "wet" and 13 spaces afterwards:
Str[0] = w = 119
str [1] = e = 101
str[10 ] = " " = 32
DE = two spaces = 32 * 10 + 32 = 352
GH = two spaces = 352
DE + GH / 381 + A = .2059
Temp = B(101) + .2059 - 32 = 69.2059

69.2069 % 128 = 69.
Looking at the table, its initial hash was indeed at 69, however, it needed 21 probes and eventually got placed in index 47.
The insertion logic is as follows:
R = 1 initially
Initial Hash = 69 initially

1) R = (5 * 1) % 256 = 5, Random step = 5/4 = 1. New index = 69 + 1 % 128 = 70 (occupied)
2) R = 5 * 5 % 256 = 25, RS = 25/4 = 6, NI = 70 + 6  = 76 (occupied)
3) etc…

I put a line of code that prints out the results of each iteration for debugging sake:

```
Probe count: 1, R: 5, Random Step: 1, Current Index: 70
Probe count: 2, R: 25, Random Step: 6, Current Index: 76
Probe count: 3, R: 125, Random Step: 31, Current Index: 107
Probe count: 4, R: 113, Random Step: 28, Current Index: 7
Probe count: 5, R: 53, Random Step: 13, Current Index: 20
Probe count: 6, R: 9, Random Step: 2, Current Index: 22
Probe count: 7, R: 45, Random Step: 11, Current Index: 33
Probe count: 8, R: 225, Random Step: 56, Current Index: 89
Probe count: 9, R: 101, Random Step: 25, Current Index: 114
Probe count: 10, R: 249, Random Step: 62, Current Index: 48
Probe count: 11, R: 221, Random Step: 55, Current Index: 103
Probe count: 12, R: 81, Random Step: 20, Current Index: 123
Probe count: 13, R: 149, Random Step: 37, Current Index: 32
Probe count: 14, R: 233, Random Step: 58, Current Index: 90
Probe count: 15, R: 141, Random Step: 35, Current Index: 125
Probe count: 16, R: 193, Random Step: 48, Current Index: 45
Probe count: 17, R: 197, Random Step: 49, Current Index: 94
Probe count: 18, R: 217, Random Step: 54, Current Index: 20
Probe count: 19, R: 61, Random Step: 15, Current Index: 35
Probe count: 20, R: 49, Random Step: 12, Current Index: 47
```

Thus, the first nonoccupied slot that the insertion function runs into is 47.

F)
There are a few key weaknesses with the hash function provided. I'll repeat it here so its easier to reference:

HA = abs{ str(2:2) + [ ( str(4:5) + str(7:8) ) / 381 + str(1:1) ] /587 – str(11) }

1) The hash function is heavily reliant on only a few characters of the hash function. Many words across English have similar letters at key points of a work, therefor only keeping in mind a few characters across a larger word would increase the total amount of collisions. A good hash function should use the entire string to increase randomness.
2) The addition and subtraction functions are weighted too heavily. If the character is too close to the ASCII value of “a”, the function will be clustered towards the top of the table, whereas if the ASCII is closer to “z”, it will be clustered towards the bottom. If the keys provided tend to start with letters closer to “a”, it will not be uniformly distributed, but rather cluster towards the top of the table.
3) The division constraints (381 and 587) don’t seem to have any relation in making sure the hash table is uniformly distributed. It would be better if they were more related to the hash table size or were smaller so that it could more uniformly distribute the keys, as larger numbers tend to cluster up the outputs. 587 is a prime number, which is generally good, due to prime numbers' nature of avoiding patterns, however, it would be better if it was coprime to the table size, so that every input could be mapped to a different slot in the table.

Upon inspection, there are some things the original hash function doesn’t implement that a great hash function uses

1) Utilization of a bitwise shift.
    a) Bitwise shifts are extremely hardware-efficient ways to multiply numbers by powers of 2. This would effectively spread out the high and low bits of the function
    b) Afterwards, you could multiply it by a prime number to randomly distribute the key across the table.
2) Using a preset value for the starting point of the HA.
    a) Starting from 0 leads to a lack of complexity, as well as issues when the HA is multiplied and divided, as multiplying and dividing by 0 is an automatic crash of the function. Utilizing addition from the starting point of 0 makes your hash function just a sum of ASCII values.
    b) Daniel J Bernstein chose the number 5381 as a random prime number that yielded good results for less collisions.

Therefore a better hash function should have the following qualities:

1) It should use every character in the string to increase randomness and avoid clustering
2) It should use bitwise operations to spread out the high and low bits of the function
3) It should use the addition of a prime number to uniformly distribute the key
4) It should be modded by the table size to fit that large number into the table while maintaining the random nature of the number generated
5) The original HA number will not be set to 0, instead being set to 5381 to better help distribution.

Here is the function I came up with:

```
int improvedHashFunction(const string& str) {
   long hash = 5381;
```

```
    for (char c : str) {
        hash = ((hash << 5) + hash) + c; // Equivalent to hash * 33 + c
    }
    return hash % TABLE_SIZE;  // Modulo table size to ensure it fits within the table
}
```

The function utilizes a for loop to iterate over each character in the string, then uses the bitwise operation to spread out the high and low bits of the function, then pulls from the string itself to make each key unique, then mods the HA to put the random number back into the table.

G) Option B vs Option C

I printed out the tables for both options, and the results are identical. This is because the algorithms themselves are consistent despite being packaged in a different way (call by value/reference). There are some differences in between I/O File redirection and main memory, however, such as speed (main memory is much faster) and the data being transient (lost as soon as the program terminates).

## C Option Hashing Code:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <array>
#include <cmath>
using namespace std;

struct Record {
    string key;
    bool isOccupied = false;
    int initialHashAddress = -1;
    int probeCount = 0;
};

const int TABLE_SIZE = 128;
Record hashTable[TABLE_SIZE];

/* legacy code for old hash function
int myAbs(int x) {
    return x < 0 ? -x : x;
}
*/

/* old hash function
int hashFunction(const string& str) {
    long long B = str[1];
    long long A = str[0];
    long long K = str[10];
    long long DE = (str[3] * 10 + str[4]) - ('0' * 11);
    long long GH = (str[6] * 10 + str[7]) - ('0' * 11);

    long long temp = B + ((DE + GH) / 381.0 + A) / 587.0 - K;
    return myAbs(static_cast<int>(temp)) % TABLE_SIZE;
}
*/

// new hash function
int hashFunction(const string& str) {
    long hash = 5381;
    for (char c : str) {
        if (!isspace(c)) { // Focus on non-whitespace characters
```

```
            hash = (((hash << 5) + hash) + c) % TABLE_SIZE;
        }
    }
    return hash;
}

array<int, 25> first25Probes;
array<int, 25> last25Probes;
int numKeysInserted = 0;

void insert(const string& key) {
    int R = 1;
    int index = hashFunction(key);
    int start_index = index;
    int probe_count = 0;

    while (true) {
        if (!hashTable[index].isOccupied) {
            hashTable[index].key = key;
            hashTable[index].isOccupied = true;
            hashTable[index].initialHashAddress = start_index;
            hashTable[index].probeCount = probe_count + 1;

            if (numKeysInserted < 25) {
                first25Probes[numKeysInserted] = probe_count + 1;
            }
            if (numKeysInserted >= 25) {
                last25Probes[numKeysInserted % 25] = probe_count + 1; // Circular buffer
            }
            numKeysInserted++;

            return;
        }


        R = (5 * R) % (int)pow(2, 8);
        int randomStep = R / 4; // p = R / 4
        index = (index + randomStep) % TABLE_SIZE; // Use randomStep
        probe_count++;

        if (index == start_index) {
            cerr << "Hash table is full" << endl;
            return;
        }
    }
}


int search(const string& key) {
    int R = 1;
    int index = hashFunction(key);
    int start_index = index;
    int probe_count = 0;
    int visited_count = 0;

    while (visited_count < TABLE_SIZE) {  // Ensure we don't loop indefinitely
        if (hashTable[index].isOccupied && hashTable[index].key == key) {
            return probe_count + 1;  // Found the key
        }

        // Update R and calculate the random step as in the insert function
        R = (5 * R) % (int)pow(2, 8);
        int randomStep = R / 4;
        index = (index + randomStep) % TABLE_SIZE;

        probe_count++;
        visited_count++;

        // If we've visited all possible slots or the current slot is not occupied
        if (!hashTable[index].isOccupied) {
            cout << "Could not find key: " << key << endl;
            return -1;
        }
    }
}




void calculateActualAverageProbes(ostream& os) {
    int totalProbes = 0;
    int count = 0;

    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i].isOccupied) {
            int probes = search(hashTable[i].key);
            if (probes != -1) {  // Only count if the key was found
                totalProbes += probes;
                count++;
            }
        }
    }

    if (count > 0) {
        os << "Actual Average Number of Probes to Find All Keys: " <<
(static_cast<double>(totalProbes) / count) << endl;
    }
    else {
        os << "No keys were found." << endl;
    }
}


void computeStats(ostream& os, const array<int, 25>& probes) {
    int minProbes = INT_MAX;
    int maxProbes = INT_MIN;
    double sum = 0;

    for (int probe : probes) {
        if (probe < minProbes) minProbes = probe;
        if (probe > maxProbes) maxProbes = probe;
        sum += probe;
    }

    os << "Minimum Probes: " << minProbes << endl;
    os << "Maximum Probes: " << maxProbes << endl;
    os << "Average Probes: " << (sum / probes.size()) << endl;

}


void printExpectedProbes(ostream& os, int n, int m) {
    double alpha = static_cast<double>(n) / m;
    double expectedProbesSuccessful = 0.5 * (1 + 1 / (1 - alpha));

    os << "Theoretical Expected Number of Probes (Linear): " << expectedProbesSuccessful <<
endl;
}


void writeResultsToFile(ofstream& resultFile) {
    if (!resultFile.is_open()) {
        cerr << "Output file is not open." << endl;
        return;
    }


    resultFile << "Hash Table Contents:" << endl;
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i].isOccupied) {
            resultFile << "Index " << i << ": Key " << hashTable[i].key
                << ", Initial Hash: " << hashTable[i].initialHashAddress
                << ", Probes: " << hashTable[i].probeCount << endl;
        }
        else {
            resultFile << "Index " << i << ": Empty" << endl;
        }
    }

    // Writing statistics for first 25 and last 25 keys
    resultFile << "Statistics for First 25 Keys:" << endl;
    computeStats(resultFile, first25Probes);
    resultFile << "Statistics for Last 25 Keys:" << endl;
    computeStats(resultFile, last25Probes);

    // Theoretical expectations
    resultFile << "Theoretical Expected Probes:" << endl;
    printExpectedProbes(resultFile, numKeysInserted, TABLE_SIZE);
    calculateActualAverageProbes(resultFile);
}


int main() {
    ifstream file("C:/Users/jonat/Documents/data/data.txt");
    if (!file) {
        cerr << "Failed to open file." << endl;
        return -1;
    }

    string line;
    while (getline(file, line)) {
        if (line.length() == 16) {
            insert(line);
        }
        else {
            cerr << "Line does not contain exactly 16 characters: [" << line << "]" << endl;
        }
    }
    file.close();

    ofstream resultFile("C:/Users/jonat/Documents/data/results.txt");
    if (!resultFile) {
        cerr << "Failed to create results file." << endl;
        return -1;
    }

    // Call the function to write all results to a file
    writeResultsToFile(resultFile);

    resultFile.close();
    return 0;
}
```

## A + B Option Code:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <array>
#include <cmath>
using namespace std;

struct Record {
    string key;
    bool isOccupied = false;
    int initialHashAddress = -1;
    int probeCount = 0;
};

const int TABLE_SIZE = 128;
Record hashTable[TABLE_SIZE];

int myAbs(int x) {
    return x < 0 ? -x : x;
}


int burrisHash(const string& str) {
    long long B = str[1];
    long long A = str[0];
    long long K = str[10];
    long long DE = (str[3] * 10 + str[4]) - ('0' * 11);
    long long GH = (str[6] * 10 + str[7]) - ('0' * 11);

    long long temp = B + ((DE + GH) / 381.0 + A) / 587.0 - K;
    return abs(static_cast<int>(temp)) % TABLE_SIZE;
}


// new hash function

void newHash(const string& str, int& HA) {
    long hash = 5381;
    for (char c : str) {
        if (!isspace(c)) {
            hash = (((hash << 5) + hash) + c) % TABLE_SIZE;
        }
    }
    HA = hash;
}


array<int, 25> first25Probes;
array<int, 25> last25Probes;
int numKeysInserted = 0;

int burrisHash(const string& str);
void newHash(const string& str, int& HA);

void insert(const string& key, bool useBurris = true) {
    int index;
    int R = 1;

    // Decide which hash function to use
    if (useBurris) {
        index = burrisHash(key);
    }
    else {
        newHash(key, index);
    }

    int start_index = index;
    int probe_count = 0;

    while (true) {
        if (!hashTable[index].isOccupied) {
            hashTable[index].key = key;
            hashTable[index].isOccupied = true;
            hashTable[index].initialHashAddress = start_index;
            hashTable[index].probeCount = probe_count + 1;

            if (numKeysInserted < 25) {
                first25Probes[numKeysInserted] = probe_count + 1;
            }
            if (numKeysInserted >= 25) {
                last25Probes[numKeysInserted % 25] = probe_count + 1; // Circular buffer
            }
            numKeysInserted++;

            return;
        }
        // Update R and calculate the next probe index
        R = (5 * R) % (int)pow(2, 8);
        int randomStep = R / 4;
        index = (index + randomStep) % TABLE_SIZE;
        probe_count++;

        if (index == start_index) {
            cerr << "Hash table is full" << endl;
            return;
        }
    }
}


int search(const string& key, bool useBurris = true) {
    int R = 1;
    int index;
    if (useBurris) {
        index = burrisHash(key);
    }
    else {
        newHash(key, index);
    }
    int start_index = index;
    int probe_count = 0;
    int visited_count = 0;

    while (visited_count < TABLE_SIZE) {  // Ensure we don't loop indefinitely
        if (hashTable[index].isOccupied && hashTable[index].key == key) {
            return probe_count + 1;  // Found the key
        }

        // Update R and calculate the random step as in the insert function
        R = (5 * R) % (int)pow(2, 8);
        int randomStep = R / 4;
        index = (index + randomStep) % TABLE_SIZE;

        probe_count++;
        visited_count++;

        // If we've visited all possible slots or the current slot is not occupied
        if (!hashTable[index].isOccupied) {
            cout << "Could not find key: " << key << endl;
            return -1;
        }
    }
}




void calculateActualAverageProbes(ostream& os) {
    int totalProbes = 0;
    int count = 0;

    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i].isOccupied) {
            int probes = search(hashTable[i].key, true); //true for burris, false for new hash
            if (probes != -1) {  // Only count if the key was found
                totalProbes += probes;
                count++;
            }
        }
    }

    if (count > 0) {
        os << "Actual Average Number of Probes to Find All Keys: " <<
(static_cast<double>(totalProbes) / count) << endl;
    }
    else {
        os << "No keys were found." << endl;
    }
}


void computeStats(ostream& os, const array<int, 25>& probes) {
    int minProbes = INT_MAX;
    int maxProbes = INT_MIN;
    double sum = 0;

    for (int probe : probes) {
        if (probe < minProbes) minProbes = probe;
        if (probe > maxProbes) maxProbes = probe;
        sum += probe;
    }

    os << "Minimum Probes: " << minProbes << endl;
    os << "Maximum Probes: " << maxProbes << endl;
    os << "Average Probes: " << (sum / probes.size()) << endl;

}


void printExpectedProbes(ostream& os, int n, int m) {
    double alpha = static_cast<double>(n) / m;
    double expectedProbesSuccessful = 0.5 * (1 + 1 / (1 - alpha));

    os << "Theoretical Expected Number of Probes (Linear): " << expectedProbesSuccessful <<
endl;
}


void writeResultsToConsole() {
    cout << "Hash Table Contents:" << endl;
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i].isOccupied) {
            cout << "Index " << i << ": Key " << hashTable[i].key
                << ", Initial Hash: " << hashTable[i].initialHashAddress
                << ", Probes: " << hashTable[i].probeCount << endl;
        }
        else {
            cout << "Index " << i << ": Empty" << endl;
        }
    }

    cout << "Statistics for First 25 Keys:" << endl;
    computeStats(cout, first25Probes);
    cout << "Statistics for Last 25 Keys:" << endl;
    computeStats(cout, last25Probes);

    cout << "Theoretical Expected Probes:" << endl;
    printExpectedProbes(cout, numKeysInserted, TABLE_SIZE);
    calculateActualAverageProbes(cout);
}


int main() {
    vector<string> keys = {
       "ABCDEFGHIJKLMNOP","1234567890123456","Aguirrie        ","Alcantara       ","Bhandari
","Carmona         ","Casper          ","Cook            ","Daniels         ",
"Nienberg        ","Paschal         ","Red             ","Salkowski       ","Zulfiqar        ","Qamruddin
","Acevedo         ","Ajose           ","Arauza          ","Buck            ","Clark           ",
"Crouch          ","Davies          ","Dugger          ","Egbe            ","Ellington       ","Farral
","Garza           ","Gurung          ","Joseph          ","Kelly           ",
"Corey           ","Adam            ","Clayton         ","Dustin          ","Robert          ","Kyle
","Scott           ","Octavio         ","Judy            ","Derek           ",
"Jeffrey         ","Jordon          ","Vinnela         ","Lisa            ","Todd            ","Veronica
","Matthew         ","Michael         ","Akhila          ","John            ",
"Charles         ","James           ","Chris           ","Wade            ","Christopher     ","Fernando
","Batbold         ","Joel            ","Fabulous        ","Misogamist      ",
"Maiden          ","Eye             ","Constriction    ","Necromancer     ","Syncopate       ","Yolk
","Afterwards      ","Person          ","Northwest       ","Irreversible    ",
"Fabricate       ","Honor           ","Staple          ","Under           ","Jutty           ","Finagle
","Cook            ","Rush            ","Wine            ","Screen          ","Perfect         ",
"mole            ","parasympathetic ","poison          ","brutalize       ","cap             ","ratiocination
","cauldron        ","prepossess      ","wince           ",
"orthodontist    ","live            ","magnetic        ", "inlet          ", "constrain      ", "marsupial      ",
"rationalize     ", "scat           ", "toluene        ", "wet            "

    };

    for (const string& key : keys) {
        insert(key, true);  // Set true to use Burris hash, false for new Hash
    }


    // Directly work with hash table in memory for result calculations
    writeResultsToConsole();  // Adapt this function to output results directly to console
    return 0;
}
```

## Tables:

## Linear table for Old Hash Function (C option)

```
Hash Table Contents:
Index 0: Key Constriction    , Initial Hash: 0, Probes: 1
Index 1: Key 1234567890123456, Initial Hash: 1, Probes: 1
Index 2: Key Northwest       , Initial Hash: 79, Probes: 52
Index 3: Key Fabricate       , Initial Hash: 65, Probes: 67
Index 4: Key Honor           , Initial Hash: 79, Probes: 54
Index 5: Key Staple          , Initial Hash: 84, Probes: 50
Index 6: Key Irreversible    , Initial Hash: 6, Probes: 1
Index 7: Key Under           , Initial Hash: 78, Probes: 58
Index 8: Key ABCDEFGHIJKLMNOP, Initial Hash: 8, Probes: 1
Index 9: Key Christopher     , Initial Hash: 9, Probes: 1
Index 10: Key Jutty           , Initial Hash: 85, Probes: 54
Index 11: Key Finagle         , Initial Hash: 73, Probes: 67
Index 12: Key Necromancer     , Initial Hash: 12, Probes: 1
Index 13: Key Cook            , Initial Hash: 79, Probes: 63
Index 14: Key Rush            , Initial Hash: 85, Probes: 58
Index 15: Key Wine            , Initial Hash: 73, Probes: 71
Index 16: Key Screen          , Initial Hash: 67, Probes: 78
Index 17: Key Perfect         , Initial Hash: 69, Probes: 77
Index 18: Key mole            , Initial Hash: 79, Probes: 68
Index 19: Key parasympathetic , Initial Hash: 6, Probes: 14
Index 20: Key poison          , Initial Hash: 79, Probes: 70
Index 21: Key brutalize       , Initial Hash: 82, Probes: 68
Index 22: Key cap             , Initial Hash: 65, Probes: 86
Index 23: Key ratiocination   , Initial Hash: 7, Probes: 17
Index 24: Key cauldron        , Initial Hash: 65, Probes: 88
Index 25: Key prepossess      , Initial Hash: 82, Probes: 72
Index 26: Key wince           , Initial Hash: 73, Probes: 82
Index 27: Key orthodontist    , Initial Hash: 0, Probes: 28
Index 28: Key live            , Initial Hash: 73, Probes: 84
Index 29: Key magnetic        , Initial Hash: 65, Probes: 93
Index 30: Key inlet           , Initial Hash: 78, Probes: 81
Index 31: Key constrain       , Initial Hash: 79, Probes: 81
Index 32: Key marsupial       , Initial Hash: 65, Probes: 96
Index 33: Key rationalize     , Initial Hash: 3, Probes: 31
Index 34: Key scat            , Initial Hash: 67, Probes: 96
Index 35: Key toluene         , Initial Hash: 79, Probes: 85
Index 36: Key wet             , Initial Hash: 69, Probes: 96
Index 37: Empty
Index 38: Empty
Index 39: Empty
Index 40: Empty
Index 41: Empty
Index 42: Empty
Index 43: Empty
Index 44: Empty
Index 45: Empty
Index 46: Empty
Index 47: Empty
Index 48: Empty
Index 49: Empty
Index 50: Empty
Index 51: Empty
Index 52: Empty
Index 53: Empty
Index 54: Empty
Index 55: Empty
Index 56: Empty
Index 57: Empty
Index 58: Empty
Index 59: Empty
Index 60: Empty
Index 61: Empty
Index 62: Empty
Index 63: Empty
Index 64: Empty
Index 65: Key Carmona         , Initial Hash: 65, Probes: 1
Index 66: Key Casper          , Initial Hash: 65, Probes: 2
Index 67: Key Daniels         , Initial Hash: 65, Probes: 3
Index 68: Key Paschal         , Initial Hash: 65, Probes: 4
Index 69: Key Red             , Initial Hash: 69, Probes: 1
Index 70: Key Salkowski       , Initial Hash: 65, Probes: 6
Index 71: Key Aguirrie        , Initial Hash: 71, Probes: 1
Index 72: Key Bhandari        , Initial Hash: 72, Probes: 1
Index 73: Key Nienberg        , Initial Hash: 73, Probes: 1
Index 74: Key Qamruddin       , Initial Hash: 65, Probes: 10
Index 75: Key Acevedo         , Initial Hash: 67, Probes: 9
Index 76: Key Alcantara       , Initial Hash: 76, Probes: 1
Index 77: Key Ajose           , Initial Hash: 74, Probes: 4
Index 78: Key Clark           , Initial Hash: 76, Probes: 3
Index 79: Key Cook            , Initial Hash: 79, Probes: 1
Index 80: Key Davies          , Initial Hash: 65, Probes: 16
Index 81: Key Egbe            , Initial Hash: 71, Probes: 11
Index 82: Key Arauza          , Initial Hash: 82, Probes: 1
Index 83: Key Crouch          , Initial Hash: 82, Probes: 2
Index 84: Key Ellington       , Initial Hash: 76, Probes: 9
Index 85: Key Zulfiqar        , Initial Hash: 85, Probes: 1
Index 86: Key Buck            , Initial Hash: 85, Probes: 2
Index 87: Key Dugger          , Initial Hash: 85, Probes: 3
Index 88: Key Farral          , Initial Hash: 65, Probes: 24
Index 89: Key Garza           , Initial Hash: 65, Probes: 25
Index 90: Key Gurung          , Initial Hash: 85, Probes: 6
Index 91: Key Joseph          , Initial Hash: 79, Probes: 13
Index 92: Key Kelly           , Initial Hash: 69, Probes: 24
Index 93: Key Corey           , Initial Hash: 79, Probes: 15
Index 94: Key Adam            , Initial Hash: 68, Probes: 27
Index 95: Key Clayton         , Initial Hash: 76, Probes: 20
Index 96: Key Dustin          , Initial Hash: 85, Probes: 12
Index 97: Key Robert          , Initial Hash: 79, Probes: 19
Index 98: Key Kyle            , Initial Hash: 89, Probes: 10
Index 99: Key Scott           , Initial Hash: 67, Probes: 33
Index 100: Key Octavio        , Initial Hash: 67, Probes: 34
Index 101: Key Judy           , Initial Hash: 85, Probes: 17
Index 102: Key Derek          , Initial Hash: 69, Probes: 34
Index 103: Key Jeffrey        , Initial Hash: 69, Probes: 35
Index 104: Key Jordon         , Initial Hash: 79, Probes: 26
Index 105: Key Vinnela        , Initial Hash: 73, Probes: 33
Index 106: Key Lisa           , Initial Hash: 73, Probes: 34
Index 107: Key Todd           , Initial Hash: 79, Probes: 29
Index 108: Key Veronica       , Initial Hash: 69, Probes: 40
Index 109: Key Matthew        , Initial Hash: 65, Probes: 45
Index 110: Key Michael        , Initial Hash: 73, Probes: 38
Index 111: Key Akhila         , Initial Hash: 75, Probes: 37
Index 112: Key John           , Initial Hash: 79, Probes: 34
Index 113: Key Charles        , Initial Hash: 72, Probes: 42
Index 114: Key James          , Initial Hash: 65, Probes: 50
Index 115: Key Chris          , Initial Hash: 72, Probes: 44
Index 116: Key Wade           , Initial Hash: 65, Probes: 52
Index 117: Key Fernando       , Initial Hash: 69, Probes: 49
Index 118: Key Batbold        , Initial Hash: 65, Probes: 54
Index 119: Key Joel           , Initial Hash: 79, Probes: 41
Index 120: Key Fabulous       , Initial Hash: 65, Probes: 56
Index 121: Key Misogamist     , Initial Hash: 73, Probes: 49
Index 122: Key Maiden         , Initial Hash: 65, Probes: 58
Index 123: Key Eye            , Initial Hash: 89, Probes: 35
Index 124: Key Syncopate      , Initial Hash: 89, Probes: 36
Index 125: Key Yolk           , Initial Hash: 79, Probes: 47
Index 126: Key Afterwards     , Initial Hash: 70, Probes: 57
Index 127: Key Person         , Initial Hash: 69, Probes: 59
```

```
Statistics for First 25 Keys:
Minimum Probes: 1
Maximum Probes: 16
Average Probes: 3.8
Statistics for Last 25 Keys:
Minimum Probes: 14
Maximum Probes: 96
Average Probes: 70
Theoretical Expected Probes:
Theoretical Expected Number of Probes (Successful): 2.78571
```

## Random Table for Old Hash Function (C option):

```
Hash Table Contents:
Index 0: Key Constriction    , Initial Hash: 0, Probes: 1
Index 1: Key 1234567890123456, Initial Hash: 1, Probes: 1
Index 2: Key Kelly           , Initial Hash: 69, Probes: 2
Index 3: Key Scott           , Initial Hash: 67, Probes: 4
Index 4: Key Fernando        , Initial Hash: 69, Probes: 4
Index 5: Key Vinnela         , Initial Hash: 73, Probes: 2
Index 6: Key Acevedo         , Initial Hash: 67, Probes: 3
Index 7: Key Misogamist      , Initial Hash: 73, Probes: 2
Index 8: Key ABCDEFGHIJKLMNOP, Initial Hash: 8, Probes: 1
Index 9: Key Crouch          , Initial Hash: 82, Probes: 2
Index 10: Key Clark          , Initial Hash: 76, Probes: 2
Index 11: Key marsupial      , Initial Hash: 65, Probes: 4
Index 12: Key Jeffrey        , Initial Hash: 69, Probes: 3
Index 13: Key Chris          , Initial Hash: 72, Probes: 3
Index 14: Key Jordon         , Initial Hash: 79, Probes: 2
Index 15: Key Dugger         , Initial Hash: 85, Probes: 2
Index 16: Key Joel           , Initial Hash: 79, Probes: 10
Index 17: Empty
Index 18: Key Matthew        , Initial Hash: 65, Probes: 3
Index 19: Key prepossess     , Initial Hash: 82, Probes: 3
Index 20: Empty
Index 21: Empty
Index 22: Key live           , Initial Hash: 73, Probes: 3
Index 23: Key poison         , Initial Hash: 79, Probes: 4
Index 24: Key cauldron       , Initial Hash: 65, Probes: 15
Index 25: Empty
Index 26: Key John           , Initial Hash: 79, Probes: 3
Index 27: Key Yolk           , Initial Hash: 79, Probes: 6
Index 28: Key Wade           , Initial Hash: 65, Probes: 4
Index 29: Empty
Index 30: Empty
Index 31: Key Fabulous        , Initial Hash: 65, Probes: 5
Index 32: Key Jutty           , Initial Hash: 85, Probes: 3
Index 33: Key wet             , Initial Hash: 69, Probes: 9
Index 34: Key rationalize     , Initial Hash: 3, Probes: 7
Index 35: Key Necromancer     , Initial Hash: 12, Probes: 4
Index 36: Key Christopher     , Initial Hash: 9, Probes: 2
Index 37: Key Under           , Initial Hash: 78, Probes: 3
Index 38: Key Irreversible    , Initial Hash: 6, Probes: 2
Index 39: Empty
Index 40: Empty
Index 41: Key mole            , Initial Hash: 79, Probes: 5
Index 42: Key parasympathetic , Initial Hash: 6, Probes: 9
Index 43: Key Rush            , Initial Hash: 85, Probes: 4
Index 44: Empty
Index 45: Empty
Index 46: Key wince           , Initial Hash: 73, Probes: 3
Index 47: Key Person          , Initial Hash: 69, Probes: 5
Index 48: Key Fabricate       , Initial Hash: 65, Probes: 6
Index 49: Key Syncopate       , Initial Hash: 89, Probes: 4
Index 50: Empty
Index 51: Empty
Index 52: Key Honor           , Initial Hash: 79, Probes: 4
Index 53: Empty
Index 54: Empty
Index 55: Empty
Index 56: Key ratiocination   , Initial Hash: 7, Probes: 2
Index 57: Key brutalize       , Initial Hash: 82, Probes: 8
Index 58: Empty
Index 59: Empty
Index 60: Key Corey           , Initial Hash: 79, Probes: 3
Index 61: Empty
Index 62: Empty
Index 63: Empty
Index 64: Empty
Index 65: Key Carmona         , Initial Hash: 65, Probes: 1
Index 66: Key Casper          , Initial Hash: 65, Probes: 2
Index 67: Key Qamruddin       , Initial Hash: 65, Probes: 2
Index 68: Key Adam            , Initial Hash: 68, Probes: 1
Index 69: Key Red             , Initial Hash: 69, Probes: 1
Index 70: Key Afterwards      , Initial Hash: 70, Probes: 1
Index 71: Key Aguirrie        , Initial Hash: 71, Probes: 1
Index 72: Key Bhandari        , Initial Hash: 72, Probes: 1
Index 73: Key Nienberg        , Initial Hash: 73, Probes: 1
Index 74: Key Ajose          , Initial Hash: 74, Probes: 1
Index 75: Key Akhila         , Initial Hash: 75, Probes: 1
Index 76: Key Alcantara      , Initial Hash: 76, Probes: 1
Index 77: Key Charles        , Initial Hash: 72, Probes: 2
Index 78: Key Salkowski      , Initial Hash: 65, Probes: 2
Index 79: Key Cook           , Initial Hash: 79, Probes: 1
Index 80: Empty
Index 81: Key scat           , Initial Hash: 67, Probes: 10
Index 82: Key Arauza         , Initial Hash: 82, Probes: 1
Index 83: Key inlet          , Initial Hash: 78, Probes: 2
Index 84: Key Staple         , Initial Hash: 84, Probes: 1
Index 85: Key Zulfiqar       , Initial Hash: 85, Probes: 1
Index 86: Empty
Index 87: Key Perfect        , Initial Hash: 69, Probes: 2
Index 88: Empty
Index 89: Key Kyle           , Initial Hash: 89, Probes: 1
Index 90: Key toluene        , Initial Hash: 79, Probes: 2
Index 91: Key Joseph         , Initial Hash: 79, Probes: 2
Index 92: Empty
Index 93: Key Paschal        , Initial Hash: 65, Probes: 2
Index 94: Key Dustin         , Initial Hash: 85, Probes: 2
Index 95: Key Cook           , Initial Hash: 79, Probes: 2
Index 96: Key Screen         , Initial Hash: 67, Probes: 2
Index 97: Empty
Index 98: Key Northwest      , Initial Hash: 79, Probes: 2
Index 99: Empty
Index 100: Key Gurung        , Initial Hash: 85, Probes: 2
Index 101: Key Derek         , Initial Hash: 69, Probes: 2
Index 102: Key Daniels       , Initial Hash: 65, Probes: 3
Index 103: Key Michael       , Initial Hash: 73, Probes: 2
Index 104: Key Judy          , Initial Hash: 85, Probes: 2
Index 105: Key orthodontist  , Initial Hash: 0, Probes: 3
Index 106: Key Egbe          , Initial Hash: 71, Probes: 2
Index 107: Key Finagle       , Initial Hash: 73, Probes: 4
Index 108: Key Batbold       , Initial Hash: 65, Probes: 3
Index 109: Key Octavio       , Initial Hash: 67, Probes: 2
Index 110: Key Buck          , Initial Hash: 85, Probes: 2
Index 111: Key Eye           , Initial Hash: 89, Probes: 5
Index 112: Key Veronica      , Initial Hash: 69, Probes: 2
Index 113: Key Todd          , Initial Hash: 79, Probes: 2
Index 114: Key Farral        , Initial Hash: 65, Probes: 2
Index 115: Key cap           , Initial Hash: 65, Probes: 5
Index 116: Key Clayton       , Initial Hash: 76, Probes: 2
Index 117: Key Wine          , Initial Hash: 73, Probes: 2
Index 118: Key Lisa            , Initial Hash: 73, Probes: 2
Index 119: Key Garza           , Initial Hash: 65, Probes: 2
Index 120: Key Maiden          , Initial Hash: 65, Probes: 2
Index 121: Empty
Index 122: Key Davies          , Initial Hash: 65, Probes: 3
Index 123: Key magnetic        , Initial Hash: 65, Probes: 6
Index 124: Key Ellington       , Initial Hash: 76, Probes: 2
Index 125: Key Robert          , Initial Hash: 79, Probes: 2
Index 126: Key constrain       , Initial Hash: 79, Probes: 8
Index 127: Key James           , Initial Hash: 65, Probes: 5
Statistics for First 25 Keys:
Minimum Probes: 1
Maximum Probes: 3
Average Probes: 1.64
Statistics for Last 25 Keys:
Minimum Probes: 2
Maximum Probes: 15
Average Probes: 4.96

Theoretical Expected Number of Probes under Random Probing: 1.94538
```

## Linear Burris Table(A + B Option):

```
Hash Table Contents:
Index 0: Key Constriction    , Initial Hash: 0, Probes: 1
Index 1: Key 1234567890123456, Initial Hash: 1, Probes: 1
Index 2: Key Northwest       , Initial Hash: 79, Probes: 52
Index 3: Key Fabricate       , Initial Hash: 65, Probes: 67
Index 4: Key Honor           , Initial Hash: 79, Probes: 54
Index 5: Key Staple          , Initial Hash: 84, Probes: 50
Index 6: Key Irreversible    , Initial Hash: 6, Probes: 1
Index 7: Key Under           , Initial Hash: 78, Probes: 58
Index 8: Key ABCDEFGHIJKLMNOP, Initial Hash: 8, Probes: 1
Index 9: Key Christopher     , Initial Hash: 9, Probes: 1
Index 10: Key Jutty          , Initial Hash: 85, Probes: 54
Index 11: Key Finagle        , Initial Hash: 73, Probes: 67
Index 12: Key Necromancer    , Initial Hash: 12, Probes: 1
Index 13: Key Cook           , Initial Hash: 79, Probes: 63
Index 14: Key Rush           , Initial Hash: 85, Probes: 58
Index 15: Key Wine           , Initial Hash: 73, Probes: 71
Index 16: Key Screen         , Initial Hash: 67, Probes: 78
Index 17: Key Perfect        , Initial Hash: 69, Probes: 77
Index 18: Key mole           , Initial Hash: 79, Probes: 68
Index 19: Key parasympathetic , Initial Hash: 6, Probes: 14
Index 20: Key poison         , Initial Hash: 79, Probes: 70
Index 21: Key brutalize      , Initial Hash: 82, Probes: 68
Index 22: Key cap            , Initial Hash: 65, Probes: 86
Index 23: Key ratiocination  , Initial Hash: 7, Probes: 17
Index 24: Key cauldron       , Initial Hash: 65, Probes: 88
Index 25: Key prepossess     , Initial Hash: 82, Probes: 72
Index 26: Key wince          , Initial Hash: 73, Probes: 82
Index 27: Key orthodontist   , Initial Hash: 0, Probes: 28
Index 28: Key live           , Initial Hash: 73, Probes: 84
Index 29: Key magnetic       , Initial Hash: 65, Probes: 93
Index 30: Key inlet          , Initial Hash: 78, Probes: 81
Index 31: Key constrain      , Initial Hash: 79, Probes: 81
Index 32: Key marsupial      , Initial Hash: 65, Probes: 96
Index 33: Key rationalize    , Initial Hash: 3, Probes: 31
Index 34: Key scat           , Initial Hash: 67, Probes: 96
Index 35: Key toluene        , Initial Hash: 79, Probes: 85
Index 36: Key wet            , Initial Hash: 69, Probes: 96
Index 37: Empty
Index 38: Empty
Index 39: Empty
Index 40: Empty
Index 41: Empty
Index 42: Empty
Index 43: Empty
Index 44: Empty
Index 45: Empty
Index 46: Empty
Index 47: Empty
Index 48: Empty
Index 49: Empty
Index 50: Empty
Index 51: Empty
Index 52: Empty
Index 53: Empty
Index 54: Empty
Index 55: Empty
Index 56: Empty
Index 57: Empty
Index 58: Empty
Index 59: Empty
Index 60: Empty
Index 61: Empty
Index 62: Empty
Index 63: Empty
Index 64: Empty
Index 65: Key Carmona        , Initial Hash: 65, Probes: 1
Index 66: Key Casper         , Initial Hash: 65, Probes: 2
Index 67: Key Daniels        , Initial Hash: 65, Probes: 3
Index 68: Key Paschal        , Initial Hash: 65, Probes: 4
Index 69: Key Red            , Initial Hash: 69, Probes: 1
Index 70: Key Salkowski      , Initial Hash: 65, Probes: 6
Index 71: Key Aguirrie       , Initial Hash: 71, Probes: 1
Index 72: Key Bhandari       , Initial Hash: 72, Probes: 1
Index 73: Key Nienberg       , Initial Hash: 73, Probes: 1
Index 74: Key Qamruddin      , Initial Hash: 65, Probes: 10
Index 75: Key Acevedo        , Initial Hash: 67, Probes: 9
Index 76: Key Alcantara      , Initial Hash: 76, Probes: 1
Index 77: Key Ajose          , Initial Hash: 74, Probes: 4
Index 78: Key Clark          , Initial Hash: 76, Probes: 3
Index 79: Key Cook           , Initial Hash: 79, Probes: 1
Index 80: Key Davies         , Initial Hash: 65, Probes: 16
Index 81: Key Egbe           , Initial Hash: 71, Probes: 11
Index 82: Key Arauza         , Initial Hash: 82, Probes: 1
Index 83: Key Crouch         , Initial Hash: 82, Probes: 2
Index 84: Key Ellington      , Initial Hash: 76, Probes: 9
Index 85: Key Zulfiqar       , Initial Hash: 85, Probes: 1
Index 86: Key Buck           , Initial Hash: 85, Probes: 2
Index 87: Key Dugger         , Initial Hash: 85, Probes: 3
Index 88: Key Farral         , Initial Hash: 65, Probes: 24
Index 89: Key Garza          , Initial Hash: 65, Probes: 25
Index 90: Key Gurung         , Initial Hash: 85, Probes: 6
Index 91: Key Joseph         , Initial Hash: 79, Probes: 13
Index 92: Key Kelly          , Initial Hash: 69, Probes: 24
Index 93: Key Corey          , Initial Hash: 79, Probes: 15
Index 94: Key Adam           , Initial Hash: 68, Probes: 27
Index 95: Key Clayton        , Initial Hash: 76, Probes: 20
Index 96: Key Dustin         , Initial Hash: 85, Probes: 12
Index 97: Key Robert         , Initial Hash: 79, Probes: 19
Index 98: Key Kyle           , Initial Hash: 89, Probes: 10
Index 99: Key Scott          , Initial Hash: 67, Probes: 33
Index 100: Key Octavio       , Initial Hash: 67, Probes: 34
Index 101: Key Judy          , Initial Hash: 85, Probes: 17
Index 102: Key Derek         , Initial Hash: 69, Probes: 34
Index 103: Key Jeffrey       , Initial Hash: 69, Probes: 35
Index 104: Key Jordon        , Initial Hash: 79, Probes: 26
Index 105: Key Vinnela       , Initial Hash: 73, Probes: 33
Index 106: Key Lisa          , Initial Hash: 73, Probes: 34
Index 107: Key Todd          , Initial Hash: 79, Probes: 29
Index 108: Key Veronica        , Initial Hash: 69, Probes: 40
Index 109: Key Matthew         , Initial Hash: 65, Probes: 45
Index 110: Key Michael         , Initial Hash: 73, Probes: 38
Index 111: Key Akhila          , Initial Hash: 75, Probes: 37
Index 112: Key John            , Initial Hash: 79, Probes: 34
Index 113: Key Charles         , Initial Hash: 72, Probes: 42
Index 114: Key James           , Initial Hash: 65, Probes: 50
Index 115: Key Chris           , Initial Hash: 72, Probes: 44
Index 116: Key Wade            , Initial Hash: 65, Probes: 52
Index 117: Key Fernando        , Initial Hash: 69, Probes: 49
Index 118: Key Batbold         , Initial Hash: 65, Probes: 54
Index 119: Key Joel            , Initial Hash: 79, Probes: 41
Index 120: Key Fabulous        , Initial Hash: 65, Probes: 56
Index 121: Key Misogamist      , Initial Hash: 73, Probes: 49
Index 122: Key Maiden          , Initial Hash: 65, Probes: 58
Index 123: Key Eye             , Initial Hash: 89, Probes: 35
Index 124: Key Syncopate       , Initial Hash: 89, Probes: 36
Index 125: Key Yolk            , Initial Hash: 79, Probes: 47
Index 126: Key Afterwards      , Initial Hash: 70, Probes: 57
Index 127: Key Person          , Initial Hash: 69, Probes: 59
Statistics for First 25 Keys:
Minimum Probes: 1
Maximum Probes: 16
Average Probes: 3.8
Statistics for Last 25 Keys:
Minimum Probes: 14
Maximum Probes: 96
Average Probes: 70
Theoretical Expected Probes:
Theoretical Expected Number of Probes (Linear): 2.78571
Actual Average Number of Probes to Find All Keys: 35.15
```

## Random Burris Table (A + B Option):

```
Index 0: Key Constriction    , Initial Hash: 0, Probes: 1
Index 1: Key 1234567890123456, Initial Hash: 1, Probes: 1
Index 2: Empty
Index 3: Key Paschal         , Initial Hash: 65, Probes: 5
Index 4: Key rationalize     , Initial Hash: 3, Probes: 2
Index 5: Key Octavio         , Initial Hash: 67, Probes: 5
Index 6: Key Irreversible    , Initial Hash: 6, Probes: 1
Index 7: Key Jeffrey         , Initial Hash: 69, Probes: 5
Index 8: Key ABCDEFGHIJKLMNOP, Initial Hash: 8, Probes: 1
Index 9: Key Christopher     , Initial Hash: 9, Probes: 1
Index 10: Key Charles        , Initial Hash: 72, Probes: 5
Index 11: Key Lisa           , Initial Hash: 73, Probes: 5
Index 12: Key Necromancer     , Initial Hash: 12, Probes: 1
Index 13: Key parasympathetic , Initial Hash: 6, Probes: 3
Index 14: Key Clayton        , Initial Hash: 76, Probes: 5
Index 15: Empty
Index 16: Key Salkowski      , Initial Hash: 65, Probes: 6
Index 17: Key Robert         , Initial Hash: 79, Probes: 5
Index 18: Key Qamruddin      , Initial Hash: 65, Probes: 7
Index 19: Empty
Index 20: Key Veronica       , Initial Hash: 69, Probes: 6
Index 21: Empty
Index 22: Key Fernando       , Initial Hash: 69, Probes: 7
Index 23: Key Dustin         , Initial Hash: 85, Probes: 5
Index 24: Key Michael        , Initial Hash: 73, Probes: 6
Index 25: Key Chris          , Initial Hash: 72, Probes: 7
Index 26: Key Misogamist     , Initial Hash: 73, Probes: 7
Index 27: Empty
Index 28: Key Wade           , Initial Hash: 65, Probes: 14
Index 29: Key Davies         , Initial Hash: 65, Probes: 8
Index 30: Key Jordon         , Initial Hash: 79, Probes: 6
Index 31: Key Fabricate      , Initial Hash: 65, Probes: 20
Index 32: Key Todd           , Initial Hash: 79, Probes: 7
Index 33: Key Person         , Initial Hash: 69, Probes: 8
Index 34: Empty
Index 35: Key prepossess     , Initial Hash: 82, Probes: 7
Index 36: Key Judy           , Initial Hash: 85, Probes: 6
Index 37: Key Finagle        , Initial Hash: 73, Probes: 8
Index 38: Key Jutty          , Initial Hash: 85, Probes: 7
Index 39: Empty
Index 40: Key toluene        , Initial Hash: 79, Probes: 23
Index 41: Key Fabulous       , Initial Hash: 65, Probes: 17
Index 42: Key Honor          , Initial Hash: 79, Probes: 14
Index 43: Key John           , Initial Hash: 79, Probes: 8
Index 44: Key Garza          , Initial Hash: 65, Probes: 11
Index 45: Key ratiocination  , Initial Hash: 7, Probes: 4
Index 46: Empty
Index 47: Key wet            , Initial Hash: 69, Probes: 21
Index 48: Key Perfect        , Initial Hash: 69, Probes: 11
Index 49: Key Rush           , Initial Hash: 85, Probes: 8
Index 50: Empty
Index 51: Empty
Index 52: Key cauldron       , Initial Hash: 65, Probes: 27
Index 53: Empty
Index 54: Empty
Index 55: Key mole           , Initial Hash: 79, Probes: 17
Index 56: Empty
Index 57: Key constrain      , Initial Hash: 79, Probes: 21
Index 58: Key Yolk           , Initial Hash: 79, Probes: 11
Index 59: Empty
Index 60: Empty
Index 61: Empty
Index 62: Empty
Index 63: Empty
Index 64: Empty
Index 65: Key Carmona        , Initial Hash: 65, Probes: 1
Index 66: Key Casper         , Initial Hash: 65, Probes: 2
Index 67: Key Acevedo        , Initial Hash: 67, Probes: 1
Index 68: Key Adam           , Initial Hash: 68, Probes: 1
Index 69: Key Red            , Initial Hash: 69, Probes: 1
Index 70: Key Kelly          , Initial Hash: 69, Probes: 2
Index 71: Key Aguirrie       , Initial Hash: 71, Probes: 1
Index 72: Key Bhandari       , Initial Hash: 72, Probes: 1
Index 73: Key Nienberg       , Initial Hash: 73, Probes: 1
Index 74: Key Ajose          , Initial Hash: 74, Probes: 1
Index 75: Key Akhila         , Initial Hash: 75, Probes: 1
Index 76: Key Alcantara      , Initial Hash: 76, Probes: 1
Index 77: Key Clark          , Initial Hash: 76, Probes: 2
Index 78: Key Egbe           , Initial Hash: 71, Probes: 3
Index 79: Key Cook           , Initial Hash: 79, Probes: 1
Index 80: Key Joseph         , Initial Hash: 79, Probes: 2
Index 81: Key orthodontist   , Initial Hash: 0, Probes: 7
Index 82: Key Arauza         , Initial Hash: 82, Probes: 1
Index 83: Key Crouch         , Initial Hash: 82, Probes: 2
Index 84: Key Staple         , Initial Hash: 84, Probes: 1
Index 85: Key Zulfiqar       , Initial Hash: 85, Probes: 1
Index 86: Key Buck           , Initial Hash: 85, Probes: 2
Index 87: Key Screen         , Initial Hash: 67, Probes: 9
Index 88: Empty
Index 89: Key Kyle           , Initial Hash: 89, Probes: 1
Index 90: Key Maiden         , Initial Hash: 65, Probes: 18
Index 91: Key magnetic       , Initial Hash: 65, Probes: 28
Index 92: Key Dugger         , Initial Hash: 85, Probes: 3
Index 93: Key Wine           , Initial Hash: 73, Probes: 9
Index 94: Key live           , Initial Hash: 73, Probes: 15
Index 95: Key marsupial      , Initial Hash: 65, Probes: 29
Index 96: Key Eye            , Initial Hash: 89, Probes: 3
Index 97: Empty
Index 98: Key inlet          , Initial Hash: 78, Probes: 9
Index 99: Key Matthew        , Initial Hash: 65, Probes: 12
Index 100: Key Cook          , Initial Hash: 79, Probes: 15
Index 101: Empty
Index 102: Empty
Index 103: Key Daniels       , Initial Hash: 65, Probes: 4
Index 104: Key poison        , Initial Hash: 79, Probes: 18
Index 105: Key Scott         , Initial Hash: 67, Probes: 4
Index 106: Empty
Index 107: Key Derek         , Initial Hash: 69, Probes: 4
Index 108: Key Afterwards    , Initial Hash: 70, Probes: 4
Index 109: Empty
Index 110: Key Farral        , Initial Hash: 65, Probes: 10
Index 111: Key Vinnela       , Initial Hash: 73, Probes: 4
Index 112: Key scat          , Initial Hash: 67, Probes: 10
Index 113: Key Northwest     , Initial Hash: 79, Probes: 12
Index 114: Key Ellington     , Initial Hash: 76, Probes: 4
Index 115: Empty
Index 116: Key Under         , Initial Hash: 78, Probes: 4
Index 117: Key Corey         , Initial Hash: 79, Probes: 4
Index 118: Key wince         , Initial Hash: 73, Probes: 10
Index 119: Key James         , Initial Hash: 65, Probes: 13
Index 120: Key brutalize     , Initial Hash: 82, Probes: 4
Index 121: Key Batbold       , Initial Hash: 65, Probes: 16
Index 122: Empty
Index 123: Key Gurung        , Initial Hash: 85, Probes: 4
Index 124: Key Joel          , Initial Hash: 79, Probes: 10
Index 125: Key cap           , Initial Hash: 65, Probes: 25
Index 126: Empty
Index 127: Key Syncopate     , Initial Hash: 89, Probes: 4
Statistics for First 25 Keys:
Minimum Probes: 1
Maximum Probes: 8
Average Probes: 2.44
Statistics for Last 25 Keys:
Minimum Probes: 2
Maximum Probes: 29
Average Probes: 13.6
Theoretical Expected Probes:
Theoretical Expected Number of Probes (Linear): 2.78571
Actual Average Number of Probes to Find All Keys: 7.22
```