Hashing

Last updated 6/23/23

These slides introduce hash concepts

Hashing

- Motivation
  - Looking for an easy way to find elements that are not inherently ordered
    - Names, passwords, ID numbers
    - E.g., ID numbers may have large gaps between them which makes it difficult to store/search them linearly
      - 12345, 27364, 79203, 92996, 15000 → array with 100,000 elements
    - E.g., Last-first names
      - Smith-joe, richenbacker-nathanial, li-ni → large char array
  - Basis for encryption

Hashing

- Basic Idea
  - Convert each item to a (hopefully unique) number and store the item in an array indexed by the unique number
  - We call the unique number the tag

data		tag
smith-joe	→	23
richenbacker-nathanial	→	66
li-ni	→	5

  - Use a 100 element array
    - In this case with 3 data items

Hashing

- Hash Function
  - Converts the data value into an integer value
    - Can limit the range of the integer using the % operation
    - Example
      - id_hash function – sum the numbers in the id and %
      - Modulo 17 for a 17 element array (hash table)

ID	sum		tag
12345 →	15	% 17 →	15
27364 →	22	% 17 →	5
79203 →	21	% 17 →	4
92996 →	35	% 17 →	1
15000 →	6	% 17 →	6

Hash Table

Tag	Data
0	
1	92996
2	
3	
4	79203
5	27364
6	15000
7	
8	
9	
10	
11	
12	
13	
14	
15	12345
16	

Hashing

- Data access
  - Is id 12345 in my data (hash table)?
    - Rehash the id number and index the hash table
      - Predictable and fast for large data sets

```c
unsigned hash_fn1(unsigned val){
   // sum 5 integers in 5-digit number
   unsigned i;
   unsigned tmp;
   tmp = 0;
   for(i = 0; i < 6; i++){
      tmp += val % 10;
      val = val / 10;
   }

   return tmp;
}// end hash_fn1
```

```c
#include <stdio.h>

unsigned hash_fn1(unsigned val);

int main(void){
    setbuf(stdout, NULL);
    printf("Hash notes\n");
    printf("Dr. Johnson\n\n");

    unsigned i;
    unsigned id;

    // build hash table - brute force
    unsigned hash_table[17];
    unsigned tag;
    tag = hash_fn1(12345) % 17;
    hash_table[tag] = 12345;
    tag = hash_fn1(27364) % 17;
    hash_table[tag] = 27364;
    tag = hash_fn1(79203) % 17;
    hash_table[tag] = 79203;
    tag = hash_fn1(92996) % 17;
    hash_table[tag] = 92996;
    tag = hash_fn1(15000) % 17;
    hash_table[tag] = 15000;
```

All un-initialized array (table) locations will have garbage in them

```c
    // print hash table
    printf("Hash Table\n");
    for(i = 0; i < 17; i++)
        printf("%u\n", hash_table[i]);
    printf("\n");

    // check for a value
    id = 12345;
    if(id == hash_table[hash_fn1(id) % 17])
        printf("%u is in the hash table\n", id);
    else
        printf("%u is NOT in the hash table\n", id);

    id = 12346;
    if(id == hash_table[hash_fn1(id) % 17])
        printf("%u is in the hash table\n", id);
    else
        printf("%u is NOT in the hash table\n", id);

    return 0;
}// end main
```

```
<terminated> (exit value: 0) Class_Project
Hash notes
Dr. Johnson

Hash Table
1518649253
92996      <-
4199136
0
79203      <-
27364      <-
15000      <-
6422476
1989987520
742706833
4294967294
6422280
1989963565
4201536
6422352
12345      <-
4201536

12345 is in the hash table
12346 is NOT in the hash table
```

Hashing

- Issues
  - Uniqueness of the tag is not guaranteed
    - Use prime number for the table (%) size
    - Choose ‘good’ hashing algorithms
    - Make the table at 2x – 3x the size of data set

    - If all else fails - there are methods to deal with this
      - out of scope for this class