CSE 12 DISCUSSION 8

Pooja Bhat

What we have seen so far:

- Arrays: Provides fast indexing- O(1). However position-based insertion and deletion is a problem.
- LinkedList: Can be expanded in O(1). What about random access though?
- Stacks and Queues: Provide O(1) insert and delete. These are ordered storage structures.
- Heap : Useful for removing highest/lowest priority item
- Binary Search Trees: Can achieve insertion, deletion and search in O(logn) if we keep the tree balanced.

All of these store data in an ordered manner, and in the process of maintaining this order, we often end up compromising efficiency.

What if we didn't care about the order of data?

E.g. I need to store a bunch of Student records, where each student has a unique ID. Which data structure is best suited for this?

HASHING - BASIC PLAN

- Basic idea: The idea of hashing is to distribute the entries (keys) across an array of buckets. Given a key, the algorithm computes an index that suggests where the entry can be found
- Hash function. A hash function is any function that can be used to map data of arbitrary size to data of fixed size. The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes.
- We are not concerned about ordering them or maintaining any order at all.
- A hash function needs to be fast, deterministic and uniform

iClicker Question

- Suppose we have a set of strings {“abc”, “def”, “ghi”, “bcc”} that we’d like to store in a table of size 5. One way to solve this is to assign “a” = 1, “b”=2, ... etc to all alphabetical characters.
- h = **sum mod 5**

What index does ghi gets stored in?

A. 3
B. 2
C. 4
D. 1
E. 0

- Suppose we have a set of strings {“abc”, “def”, “ghi”, “bcc”} that we’d like to store in a table of size 5. One way to solve this is to assign “a” = 1, “b”=2, ... etc to all alphabetical characters.
- h = **sum mod 5**
- “abc” = 1 + 2 + 3=6, “def” = 4 + 5 + 6=15 , “ghi” = 7 + 8 + 9=24 “bcc” = 2+3+3=8
- “abc” in 6 mod 5 = 1, “def” in 15 mod 5 = 0, and “ghi” in 24 mod 5 = 4 and so on

0	1	2	3	4
def	abc		bcc	ghi

iClicker Question

Consider a table of size 8. Let the hash function be h = key mod 6. The problem with this hash function is?

A. It is not deterministic
B. It is not uniform
C. It is not fast
D. The hash function looks good

iClicker Question

Consider a table of size 8. Let the hash function be h = key mod 6. The problem with this hash function is?

A. It is not deterministic
B. It is not uniform
C. It is not fast
D. The hash function looks good

Answer: It is not uniform! Why?

iClicker Question

Consider a table of size 8 and a hash function h = (key + random.nextInt( )) mod 8

Problem?

A. It is not deterministic
B. It is not uniform
C. It does not generate integer index
D. It is not fast
E. No problem at all

iClicker Question

Consider a table of size 8 and a hash function h = (key + random.nextInt( )) mod 8

Problem?

A. It is not deterministic
B. It is not uniform
C. It does not generate integer index
D. It is not fast
E. No problem at all

Answer: It is not deterministic

- Hash function should be fast: No point of achieving O(1) time if it takes too long to generate the hash
- Hash function should be deterministic: If I try to lookup ‘abc’ again, the hash function should return 1 as the index
- Hash function should be uniform: Should not prefer any one index over the other.

Issues we need to deal with:

1. What happens if we insert cba?
2. What happens if we keep inserting strings? When do we decide to expand the table?

Load factor

- Load factor = N/M where N = number of keys and M = number of buckets
- As the load factor grows larger, the hash table becomes slower, and it may even fail to work (depending on the method used).
- The expected constant time property of a hash table assumes that the load factor is kept below some bound. For a fixed number of buckets, the time for a lookup grows with the number of entries and therefore the desired constant time is not achieved.
- Solution: Rehashing
- As load factor goes above a threshold, double the array size and **rehash keys into buckets.**

iClicker Question

0	1	2	3	4
def	abc		bcc	ghi

The load factor for the above table is:

A. 1/5
B. 4/5
C. 5/4
D. 1/4

iClicker Question

Which of the following best describes a collision?

A. Two entries with different keys and different hash value
B. Two entries with the exact same key.
C. Two entries with different keys have the same exact hash value.
D. Two entries with the exact same key have different hash values.

iClicker Question

Which of the following best describes a collision?

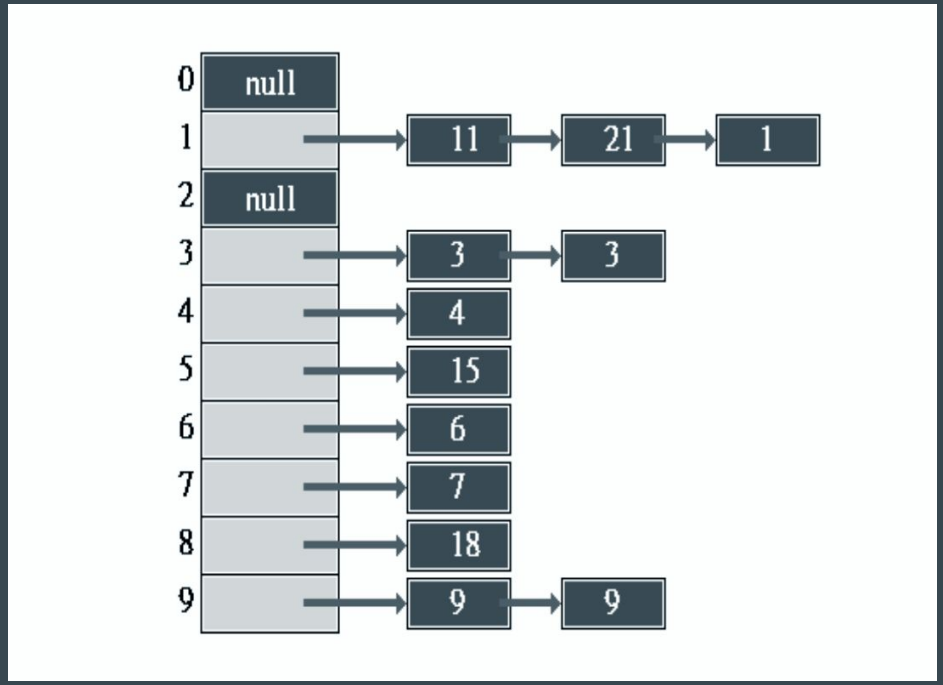
A. Two entries with different keys and different hash value
B. Two entries with the exact same key.
C. Two entries with different keys have the same exact hash value.
D. Two entries with the exact same key have different hash values.

Answer: C Two entries with different keys have the same exact hash value.

Hashing Revise: Key principles

1. A good hash function
2. Collision Resolution: “abc”, “bac” etc will end up with the same value for the sum and hence the key. **Collision**: Two distinct keys hashing to same index.
   We need to deal with collisions efficiently. In HW8, we will be using Chaining to resolve collisions.
3. Rehash: When to rehash? In HW8, rehash when load factor > 2/3

COLLISION RESOLUTION - CHAINING



- Cost is proportional to length of list.
- Best case = N / M.
- Worst case: all keys hash to same list.

iClicker Question

Consider a table of size 8. Hash the following keys using chaining. H = k mod M

36, 6, 10, 22, 15, 18, 26, 42, 47

What is the number of collisions that occur?

A. 4
B. 5
C. 6
D. 3

0	1	2	3	4	5	6	7

HW8 Part 1: DOUBLE HASHING EXAMPLE
Write your answer to this part in a neat Table format!

Let h1(k) = k mod 13 and h2(k) = 1 + (k mod 11)
(Why is it 1+ (k mod 11) and not just (k mod 11)?)

Let h(k,i) = (h1(k) + i *h2(k)) mod size

Consider a hash table as follows: Insert 14 to the table

0	1	2	3	4	5	6	7	8	9	10	11	12
	79			17	98		72		9			

HW8 Part 1: DOUBLE HASHING

0	1	2	3	4	5	6	7	8	9	10	11	12
	79			17	98		72		9			

Probe (i) 0: h(14,0) = 14 mod 13 = 1

Probe 1: h2(14) = 1 + (14 mod 11)) = 4

h(14,1) = (h1 (14) + 1 * h2(14)) mod 13

= (1 + 1*4) mod 13 = 5

Probe 2: h(14,2) = (1 + 2*4) mod 13 = 9

Probe 3: h(14,3) = (1 + 3*4)mod 13 = 0 Yayy!!

HW8 - Part 2: Implement a Hash Table

1. You need an array of LinkedList that holds strings.
   a. private LinkedList<String>[] hashTable;
2. You need to choose a good hash function of your choice
3. Collisions are resolved by chaining
4. Load factor limit = ⅔.
5. You need to calculate load factor (N/M) upon every insertion, and if it goes above ⅔, then expand the table and **rehash the items.**
6. Do not forget to recompute the indices after resizing
7. You can choose any hash function of your choice, even a simple sum%table_size will do.

Methods in HashTable

1. insert(word) : Insert ‘word’ into the table. Do not insert duplicates.
    a. index = hash_function(word, table_size)
    b. If load_factor>⅔, expand and rehash
    c. Return true for successful insert, false if item was already present
    d. Beware: If rehash isn’t properly implemented, then your hash table might not work as expected.
2. lookup(word): Search for ‘word’ in the hash table
    a. Return true if ‘word’ is found, false otherwise.
    b. How and where would you search for ‘word’?
3. delete(word): Remove ‘word’ from the hash table
    a. True if value was found and deleted, false otherwise
    b. Again, how and where would you search?
4. printTable(): follow the format in the writeup. If empty, print nothing.
5. getSize( ): Return the number of items in hash table.

iClicker Question

There are several factors that affect the efficiency of lookup operations in a hash table that uses open addressing techniques. Which of the following is one of those factors?

A. Number of elements stored in the hash table
B. Number of buckets in the hash table
C. Quality of the hash function
D. All of the above

Part 3 - SpellChecker

- In this part you will develop a SpellChecker Algorithm in SpellChecker.java
- Your program will take 2 command-line arguments: a dictionary file and an input file.
- You must first load all the words from the dictionary into the hash table
- Then your program will read a list of words from another input file, the words can be any case, upper and lower (so what do you need to do first before you check?)
- If a word is found in the dictionary, your program should produce “ok” output
- If the word is not found, you should generate suggested corrections and write them, together with the original word (converted to lowercase), as a single output line. (see below)
- If no suggestions can be found, then just print “not found”.

Generating Corrections for Pol

You must check for five possible errors:

1) a wrong letter ("pod"),
    a) In general, you have (26* n) possibilities, where n is the length of the word.
2) an inserted letter ("pogl"),
    a) In general, you have (26*(n+1)) possibilities.
3) a deleted letter ("po"),
    a) In general, you have n possibilities, where n is the length of the word
4) a pair of adjacent transposed letters ("plo").
    a) In general, you have (n-1) possibilities.
5) every pair of strings that can be made by inserting a space into the word. For example, (“p ol”)
    a) In general, you have (n-1) pairs of words; make sure to check separately in the dictionary for each word in the pair.

Implementation Tips

- Develop helper methods for each type of correction to clean up your code
- Test one type of correction before starting on the next
- Your program shouldn’t take too long to load the dictionary. If it does, then you have a very inefficient hash table
- Test all given inputs and come up with your own inputs.

Some common problems

- Find if a string contains all unique characters.
  - What if you cannot use additional O(n) space?
- Given two strings A and B, find if one is a permutation of the other.
  - abb and bba - true
  - abcd and acdb - true
  - acc and ccb - false
- Given a list, find all the pairs whose difference is k
  - Example: List is { 1, 7, 5, 9, 2, 12, 3} and k =2
  - Output Pairs: (1,3), (3,5), (5,7), (7,9)
- You are given a list of n-1 integers and these integers are in the range of 1 to n. There are no duplicates in list. One of the integers is missing in the list. Find the missing number.
  - Hint: Do you even need a hash table?