Week 9

PQ, (and a little bit of hashing)

Qn 1 - Quadratic Probing

a) Given a hash table of size 7 with h(x) = x % 7, we insert these elements in the order 5, 12, 19, 26, 2. What does the resulting hash table look like?

```
for i in 0..m:
  // increment by squares instead
  if buckets[hash(x) + i * i % m] is empty:
    insert x into this bucket
    break
```

Hash Table!

0	
1	
2	
3	
4	
5	5
6	

h(x) = x % 7, m = 7

```
for i in 0..m:
  // increment by squares instead
  if buckets[hash(x) + i * i % m] is empty:
    insert x into this bucket
    break
```

Hash Table!

0	
1	
2	
3	
4	
5	5
6	

h(x) = x % 7, m = 7

```
for i in 0..m:
  // increment by squares instead
  if buckets[hash(x) + i * i % m] is empty:
    insert x into this bucket
    break
```

How do we insert 12?

12 % 7 = 5 (collision!)

So 5 + 1 * 1 % 7 = 6

Hash Table!

0	
1	
2	
3	
4	
5	5
6	12

h(x) = x % 7, m = 7

```
for i in 0..m:
  // increment by squares instead
  if buckets[hash(x) + i * i % m] is empty:
    insert x into this bucket
    break
```

How do we insert 12?

12 % 7 = 5 (collision!)

So 5 + 1 * 1 % 7 = 6

Hash Table!

0	26
1	(empty)
2	19
3	2
4	(empty)
5	5
6	12

h(x) = x % 7, m = 7

```
for i in 0..m:
  // increment by squares instead
  if buckets[hash(x) + i * i % m] is empty:
    insert x into this bucket
    break
```

Insert remaining elements

Qn 1 - Quadratic Probing

b) Delete 12, then 5. What does the resulting table look like?

Hash Table!

0	26
1	(empty)
2	19
3	2
4	(empty)
5	5 (deleted)
6	12 (deleted)

Tombstone values! We cannot just remove it from the table. If we do, search fails, we can never find 19 or 26.

Qn 1 - Quadratic Probing

c) Construct a case where quadratic probing fails, even if the table is not empty

Hash Table!

0	occupied
1	occupied
2	X

Table of size 3, with buckets 0 and 1 being occupied and hash(x) = 0

(0 + 2 * 2) % 3 = 1

Qn 2 - Union / Intersection of Sets

Given 2 finite sets A and B, how can you efficiently find the intersection and union of two sets?

Intersection

Intersection is relatively simple:

- Hash all the items for one set (e.g A)
- Check each element of B to see if its inside A
- Overall expected runtime = O (|A| + |B|)

Union

Hash all the elements of A and B into the same table, then run through the table and don’t output duplicates.

- In chaining, have to run through the linked list in the bucket to check for duplicates
- In open addressing, have to go through the probes until encounter null.
- But in both cases expected cost is O(|A| + |B|). You insert all the elements and then you run through the whole table once.

Qn 3 - funny array problem

Given an array of n integers, and a value k. Decide if there is a contiguous sub-array whose average value is k.

Solution - naive

Try every pair of start and end points (basically try every sub-array) and compute the average. Return true once value is equal to k. Else return false.

Solution - optimal

1. Subtract every element of the array by k.
2. Compute the prefix sum array

Example where k = 6

1) arr = [1, 3, 2, 5, 7, 20]
2) o_array = [-5, -3, -4, -1, 1, 14]
3) c_array = [-5, -8, -12, -13, -12, 2]

Solution? To find i > j such that c_array[i] - c_array[j] = 0 **(???)**

Solution - optimal

1) arr = [1, 3, 2, 5, 7, 20]
2) o_array = [-5, -3, -4, -1, 1, 14]
3) c_array = [-5, -8, -12, -13, -12, 2]

Solution? To find i > j such that c_array[i] - c_array[j] = 0 **(draw on whiteboard)**

Lets think about what c_array[i] - c_array[j] represents (where i > j)

- **Sum of o_array from index 0 to i - Sum of o_array from index 0 to j**
- **Sum of o_array = Sum of arr - k**
- **Hence, it is equivalent to the sum of arr[r] - k from index i+1 to j**

Solution - optimal

1) arr = [1, 3, 2, 5, 7, 20]
2) o_array = [-5, -3, -4, -1, 1, 14]
3) c_array = [-5, -8, -12, -13, -12, 2]

Solution? To find i > j such that c_array[i] - c_array[j] = 0 **(???)**

Lets think about what c_array[i] - c_array[j] represents (where i > j)

- **Hence, it is equivalent to the sum of arr[r] - k from index i+1 to j**
- **By equating this sum to 0, we are trying to find if (j - i) * k = sum of arr[r] from i + 1 to j (which is essentially finding if the average of the subarray is equivalent to k)**

Solution - optimal

1) arr = [1, 3, 2, 5, 7, 20]
2) o_array = [-5, -3, -4, -1, 1, 14]
3) c_array = [-5, -8, -12, -13, -12, 2]

Solution? To find i > j such that c_array[i] - c_array[j] = 0 **(???)**

Notice that if we find two distinct positions in the prefix sum array that have the same value, then we know that the corresponding subarray has average value k (its like the k that you minus away each time was offset by the input values because their average value is k)

So, to find duplicate values, use a hashtable! If got collision = true.

Qn 4 - Priority Queue

Given a data set containing n unique elements, we want to know the top k valued elements

a) Implement getKLargest() better than O(nlogn)

Qn 4 - Priority Queue

Given a data set containing n unique elements, we want to know the top k valued elements

a) Implement getKLargest() better than O(nlogn)

Simple! We just QuickSelect the k’th largest element in O(n) time. Everything on the right are the top k elements.

Qn 4 - Priority Queue

Given a data set containing n unique elements, we want to know the top k valued elements

b) Support insertNext(x) in O(log k) time, and getKLargest() in O(k) time.

KEY IDEA: The bottom (n - k) elements are useless! We don’t need them to support these two queries.

Solution - optimal

We create a min heap of size <= k.

- If size < k, keep inserting elements
- If size = k and we call insert, check x against the smallest of the elements we have stored (which is the root).
  - If it is smaller, discard x.
  - If it is larger, insert x and discard the min (Extract-min)
- The heap itself stores the k largest elements and insert operations are upper bounded by O(log k).

Qn 5 - Stack 2 Queue

How to implement a stack using 2 queues?

a) Design an algorithm to enqueue (push) and dequeue (pop) from the queue using nothing but 2 stacks (labeled S1 and S2)

Qn 5 - Stack 2 Queue

How to implement a stack using 2 queues?

a) Design an algorithm to enqueue (push) and dequeue (pop) from the queue using nothing but 2 stacks (labeled S1 and S2)

Solution:

1. Push(x) = Push x onto S1
2. Pop(x) = If S2 is not empty, pop from S2. If S2 is empty, repeatedly pop from S1 and push into S2 (and the last element pop in S1 is ‘dequeued’)

Essentially if S2 is empty, we just reverse the order of S1 by popping the elements and pushing it into S2. So LIFO becomes FIFO.

Qn 5 - Stack 2 Queue

How to implement a stack using 2 queues?

b) Worst case and amortised case for each operation.

Qn 5 - Stack 2 Queue

How to implement a stack using 2 queues?

b) Worst case and amortised case for each operation.

Worst case for push: O(1)
Worst case for pop/dequeue: O(n)

What about amortised cost? O(1) for both operations. How? We use the Accounting Method!

Qn 5 - Stack 2 Queue

How to implement a queue using 2 stacks?

Accounting method:

- Suppose each time you push an element, you pay $2 instead of $1 to the bank.
- When we transfer an element from S1 to S2, or pop from S2, each of these operations cost $1.
- When pop is called and S2 is empty, we have at least **$2k** in the bank, where **k is the number of inserted elements in S1**.
- Enough to pay for the transferring cost = $k and the remaining $k used to pop from S2.
- So any sequence of t inserts and pops costs O(t) and not O(t^2) time.

Qn 6 - Min Queue

Implement a queue that supports the following operations:

- **push/enqueue**
- **pop/dequeue**
- **getMin: returns minimum value stored in the queue**

Any sequence of t operations runs in O(t) time (i.e total cost of t operations is O(t).

Qn 6 - Min Queue

Simplest solution actually uses 2 stacks.

Idea: We store pairs (x, m), where m is the minimum value of the stack. This way, for getMin all we have to do is to peek at the top of the stack and output m.

To insert x into an empty stack, insert (x, x). Else run getMin to obtain m and push (x, min (m, x)) onto the stack.

To pop, just remove the pair.

How to implement this using a queue? Use the 2 stack solution presented earlier!