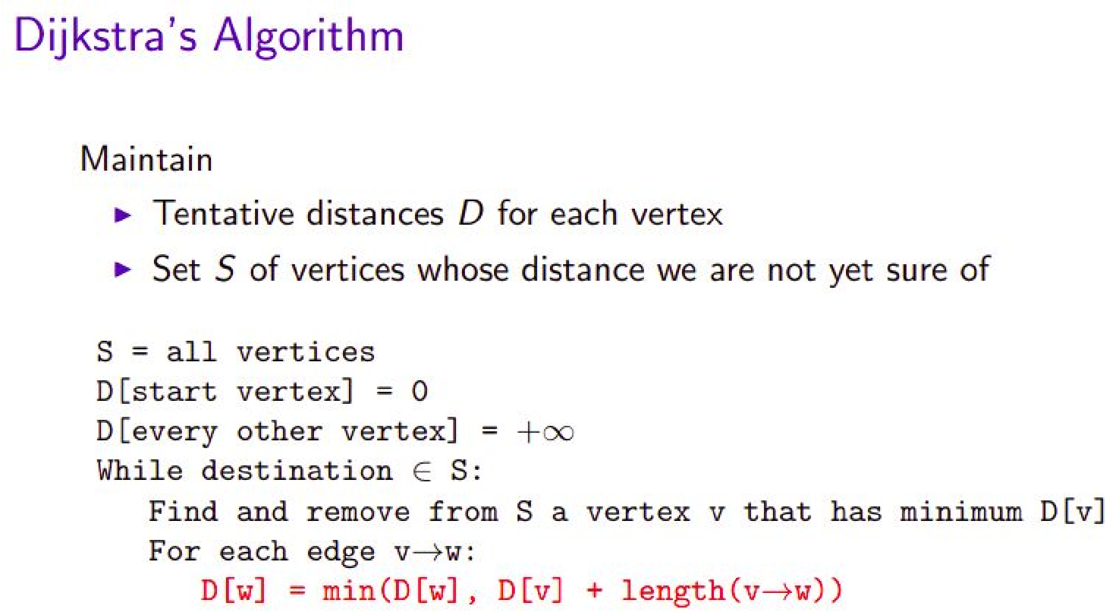
CS261P Data Structures, Spring 2020
Worksheet (with solutions) - 28th April 2020
TA: Aftab Hussain
University of California, Irvine

Q.1. Name three data structure operations in the following algorithm (from Prof. Kevin's slide deck):



Dijkstra's Algorithm

Maintain

- Tentative distances D for each vertex
- Set S of vertices whose distance we are not yet sure of

```
S = all vertices
D[start vertex] = 0
D[every other vertex] = +∞
While destination ∈ S:
   Find and remove from S a vertex v that has minimum D[v]
   For each edge v→w:
      D[w] = min(D[w], D[v] + length(v→w))
```

A.1. Finding and removing vertex with minimum D[v], and updating D[w]. (D being the concerned data structure).

Q.2. Describe the Decorator Pattern in the context of describing data structures, with an example.

A.2. An object oriented programming pattern which we use for our data structure descriptions, where we use structures that consist of objects, and we allow users to add elements of information to that structure (without affecting existing elements of information). The structure may end up having a variety of data types.
Ref. 1-18 notes1.pdf, from 09:08 onwards Lecture 3 - April 7 Video

Q.3. How does the Decorator Pattern help you implement the data structure of a line segment, for the segment crossing detection algorithm?

A.3. You can augment coordinate information of a line segment to each node of the binary search tree of the graph.

Ref. 11:03 onwards Lecture 3 - April 7 Video

Q.4. Write down the full form of LIFO. Name a data structure that works following LIFO.

A.4. Last-in-first-out. Stack

Q.5. From which side of a FIFO queue of elements do you remove/dequeue elements?

A.5. Front

Q.6. What is a deque?

A.6. Stack and queue hybrid - Ref. 2-3, notes2.pdf

Q.7. What is the amortized time of a data structure operation?

A.7. Ref. 58:55 onwards onwards Lecture 3 - April 7 Video

Q.8. Which analysis gives you more flexibility in estimating the time of a data structure operation?

A.8. Amortized analysis

Q.9. What is the worst case time complexity of incrementing the n-bit counter algorithm?

A.9. O(n)
Ref. 68:00 onwards Lecture 3 - April 7 Video

Q.10. What is the time complexity of incrementing the n-bit counter algorithm if you use a probabilistic analysis technique?

A.10. O(1)
Ref. 70:00 onwards Lecture 3 - April 7 Video.

Q.11. What is the name of the type of analysis you used in question 10, and why is it not accurate or dependable?

A.11. Average time complexity. It uses a probabilistic approach, wherein it assumes each sequence to be equally probable, which is unlikely.

Q.12. What's the result of the following operation: 1011 XOR 1001?

A.12. 0010

Q.13. What is the name of the C++ template class used for implementing Dynamic Arrays?

A.13. Vector (ArrayList in Java, and list in Python)

Q.14. Name the fundamental operations of a dictionary data structure.

A.14. Search(key), Set(key, value), Delete(key)

Q.15. In order to map key-value pairs, a hashing data structure implementation utilizes a hash table and a _______ ?

A.15. hashing function

Q.16. What property must we maintain when adding elements to a hash table?

A.16. The size of the hash table, N, should be greater than the number of key-value pairs, n, stored inside the hash table.

Q.17. What do we do in order to maintain the property in Q.16?

A.17. Resize the hash table.

Q.18. What you do in Q.17 would never involve changing the hash function. Correct?

A.18. No
Ref. 18:00 onwards Lecture 5 - April 14 Video.

Q.19. Write down the steps for implementing a common hashing function that maps a value, say a string, to an integer.

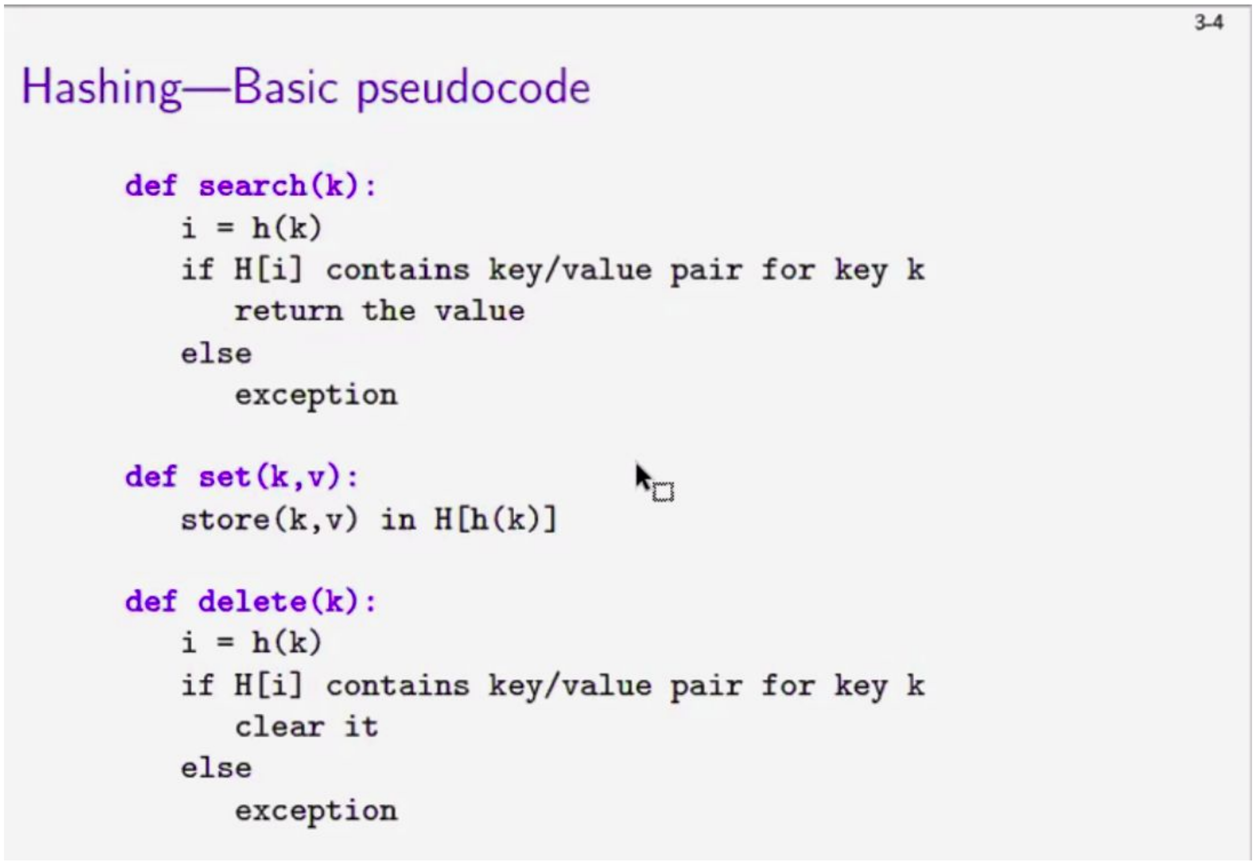
A. Ref. 23:00 onwards Lecture 5 - April 14 Video.

Q.20. Say A and B are two integers, what is the highest possible value of A modulo B?

A.20. B - 1

Q.21. Write the basic pseudocode for the implementing the operations in Q.14. by using hashing.

A.21.



3-4

Hashing—Basic pseudocode

```
def search(k):
   i = h(k)
   if H[i] contains key/value pair for key k
      return the value
   else
      exception

def set(k,v):
   store(k,v) in H[h(k)]

def delete(k):
   i = h(k)
   if H[i] contains key/value pair for key k
      clear it
   else
      exception
```

Ref. notes3.pdf

Q.22. What is collision in hashing? Name some hashing techniques that address this problem.

A.22. Multiple keys may map to the same value. Hash chaining, linear probing, and cuckoo hashing.

Q.23. Very briefly describe what each of the techniques in Q.22 do differently to address the problem of collision.

A.23. In hash chaining, each hash table key entry points to an array list of values, where each value of the array list has the same key.
In linear probing, during inserting a key-value pair, if we find that our desired slot is already occupied, we place the pair in the next available spot.

In cuckoo hashing, we use two tables in such a way that when there is a collision in one table, we eject the colliding value from that table and push it to the other table; the same approach is applied to the other table as well, iteratively.
Ref. 31:13 onwards Lecture 5 - April 14 Video.

Q.24. In hash chaining, each entry of the hash table consists of a collection of key-value pairs. We could use a vector (e.g. in C++) or an array list to implement those collections. What other data structure could we use to implement them?

A.24. A linked list.
Ref. 36:00 onwards Lecture 5 - April 14 Video.

Q.25. Name two disadvantages of hash chaining.

A.25. More time is required to scan a collection, and more space is needed to store those collections.

Q.26. In hash chaining, how can we ensure a set(k,v) operation to be always O(1)?

A.26. If we can programmatically ensure that no key can have more than one value. (In which case, the hashing technique effectively behaves like basic hashing).
Ref. 40:50 onwards Lecture 5 - April 14 Video.

Q.27. Name one advantage of linear probing over hash chaining.

A.27. It is more space efficient.
Ref. 69:00 onwards Lecture 5 - April 14 Video.

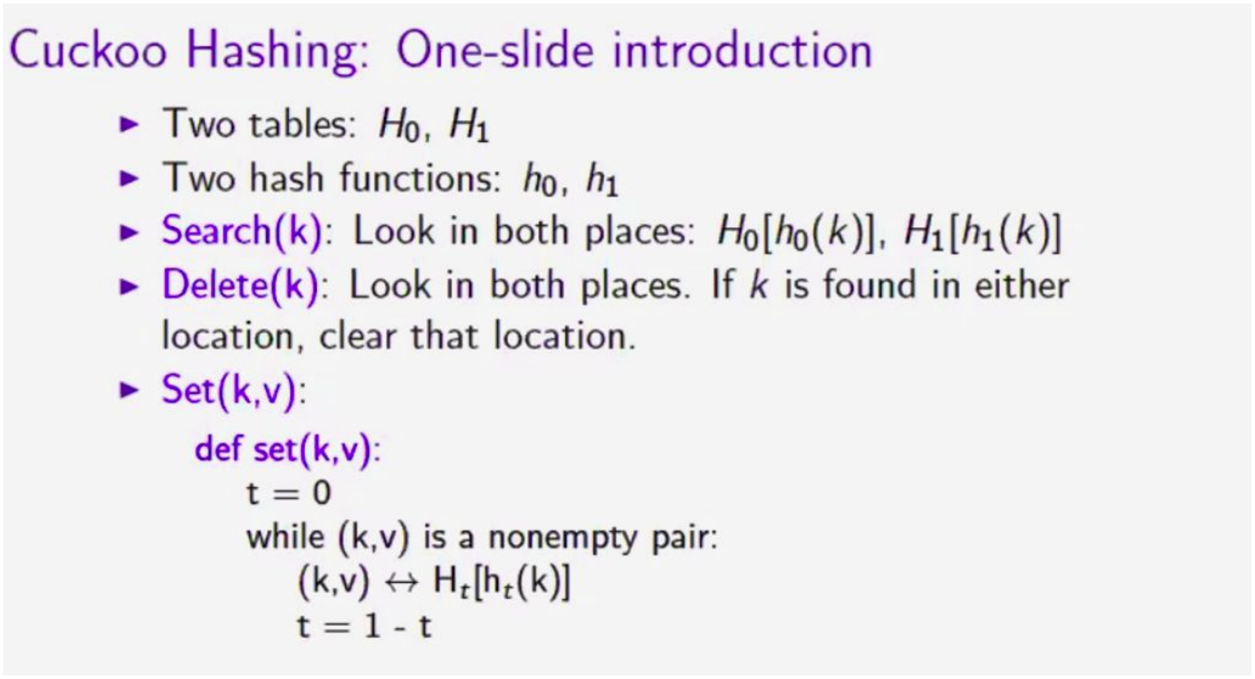
Q.28. What effect does a load factor of nearly 1 have on a set operation in a hash table implemented by linear probing?

A.28. Ref. 75:00 onwards Lecture 5 - April 14 Video.

Q.29. Name and describe two alternatives to Linear Probing.

A.29. Quadratic Probing and Double Hashing.
Ref. 78:50 onwards Lecture 5 - April 14 Video for descriptions.

Q.30. What's the role of the last line of the pseudocode below (from Prof. Kevin's slide deck)?



Cuckoo Hashing: One-slide introduction

- Two tables: H_0, H_1
- Two hash functions: h_0, h_1
- Search(k): Look in both places: $H_0[h_0(k)]$, $H_1[h_1(k)]$
- Delete(k): Look in both places. If k is found in either location, clear that location.
- Set(k,v):

```
def set(k,v):
    t = 0
    while (k,v) is a nonempty pair:
        (k,v) $\leftrightarrow$ $H_t[h_t(k)]$
        t = 1 - t
```

A.30. The line alternates the value of t between 1, 0 and thereby allows you to switch between the two hashing algorithms.
Ref. 60:00 onwards Lecture 6 - April 16 Video.

Q.31. You have a scenario where you have a dictionary that is built up once and you don't modify it much. Which hashing technique would you use for using (modifying/searching) this dictionary and why?

A.31. Cuckoo hashing. The reason being that O(1) time is guaranteed for searching, and you don't have to make modifications a lot, which could get expensive with cuckoo hashing, especially when we need to rebuild the hash functions as a result of iterating over the while loop (see pseudocode in Q.31 above).
Ref. 64:50 onwards Lecture 6 - April 16 Video.