Introduction to Algorithms: 6.006
Massachusetts Institute of Technology March 14, 2017
Professors Constantinos Daskalakis, Piotr Indyk, and Steve Smith Problem Set 3

Problem Set 3

All parts are due on April 6, 2017 at 11:59PM. Please download the .zip archive for this problem set. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Last, but not least, take a look at the collaboration policy outlined in the handout for this course.

Part A

Problem 3-1. [10 points] **Rainbow Graphs**

Let G be a connected, unweighted, undirected graph with V vertices and E edges. Given two vertices s and t, answer the following shortest-path questions:

(a) [3 points] If each **vertex** is colored with one of k colors $c_1, c_2, \ldots c_k$, compute the shortest path between s and t such that no two consecutive vertices are the same color. Your algorithm should run in $O(V + E)$.

(b) [7 points] If each each **edge** is colored with one of k colors $c_1, c_2, \ldots c_k$, compute the shortest path between s and t such that no two consecutive edges are the same color. Your algorithm should run in $O(k(V + E))$.

Problem 3-2. [20 points] **Graphrica**

The country of Graphrica (please excuse the pun) consists of V *cities* $c_1, c_2, \ldots c_V$ and E bidirectional *roads* $r_1, r_2, \ldots r_E$ between these cities. The Graphrican government is concerned that there doesnt exist a path between every pair of cities: to solve this problem, they would like to add $0 < k \leq V$ roads to maximize the number of pairs of reachable cities. Can you help the government figure out where to build these roads?

(a) [5 points] What if the Graphrican government would only like to build one road ($k = 1$)? Design an $O(V + E)$ algorithm to figure out where to build this road.

(b) [15 points] Now, the Graphrican government would like to construct k roads. Again, design an $O(V + E)$ algorithm to determine where to build these roads. **Prove that your method is correct.**

Hint: Try proving your algorithm is correct for $k = 2$, and generalize.

Problem 3-3. [15 points] **Unicycles!**

Consider a connected, undirected, positive-weighted graph G with V vertices and E edges. Given that G contains **exactly one cycle**, find the shortest path between two given vertices s and t in $O(V)$ time.

Problem 3-4. [30 points] **Hash Money**

In this problem, you will investigate how Python hash tables (`dict` objects) are implemented. Write some code and experiment with it to figure out the following problems. For each part, your writeup should consist of the following:

- Any relevant code you used to answer the question. We would appreciate it if you formatted your code nicely: consider using the `Verbatim` or `minted` modules.
- Any relevant outputs from your code.
- A brief explanation of your code and output, and how you used it to figured out the answer.

Extra credit may be given for particularly thorough or interesting writeups.

These questions are more open-ended than usual 6.006 problems: there might be multiple approaches to a problem. **If you're having trouble, look it up online before asking on Piazza!** Stack Overflow is your friend.

Tips/Hints:

- For the sake of consistency, please use Python 3 for the following questions.
- If you are having trouble solving these on your operating system, consider using an Athena machine.
- Python uses open addressing to implement its hash table.
- To find the size of an object `x` in bytes, use `sys.getsizeof(x)`.
- Note that a Python dictionary consists of multiple components, not just the array of entries. For that reason, there's some memory overhead: you can either choose to account for this overhead, or simply ignore it during your computations. Either way, you should still be able to solve (b) and (c).
- Use the `time` module to measure times. Note that because operations like dictionary insertion are extremely fast, your measures will be more accurate if you measure multiple operations at once (for example, measuring the time it takes to insert 1000 elements).

(a) [2 points] What is the size of a Python dictionary entry, in bytes? You may find `github.com/python/cpython/blob/master/Objects/dict-common.h` helpful.

Hint: The size of a pointer is 64-bits if you have a 64-bit system, and 32-bits if you have a 32-bit system. In addition, the `Py_hash_t` type is equivalent to the C++ `ssize_t` type, which is usually 8 bytes.

(b) [2 points] Recall that when the load factor α is reached, the size of a hash table increases. In other words, if you have inserted n keys into the table, and the table can hold m entries, the size increases when $n/m = \alpha$. In a Python dictionary, by how much does the table size increase when this happens?

(c) [5 points] What is the load factor of a Python dictionary?

(d) [2 points] Does it take longer to insert keys when the load factor is reached? Why exactly does this happen?

(e) [4 points] What happens if there are hash collisions in a table? Imagine we have the following code, where we define a class `C` and its corresponding hash function (in Python, you can override the default hash function by implementing `__hash__`).

```
import random

class C(object):
    def __init__(self, n):
        self.hash = random.randint(0, n)
    def __hash__(self):
        return self.hash

n = 100
d = dict()
x = C(n)
# As long as a value is hashable, it can be used as the
# key of a dictionary.
d[x] = random.random()
```

The value of n determines the frequency of collisions for the `C` object. When $n = 1$, for example, every object has the same hash. However, when $n = 100000$, collisions are much less likely. How does the value of n affect the times it takes to insert k elements in a Python dictionary? Why does this happen? **Include a graph to support your answer.**

(f) [4 points] How might the following hash function lead to problems? Provide at least **two** examples of problematic code (involving dictionaries), and explain what causes the error.

```
import random

class D(object):
    def __init__(self, n):
        self.n = n
    def __hash__(self):
        return random.randint(0, self.n)
```

(g) [4 points] In class, we discussed the adjacency list and adjacency matrix approaches to representing a graph. How could you use Python `set` objects to implement a better

approach, combining the best features of both representations? Provide a brief runtime analysis.

Note: A python `set` object is essentially a dictionary without any values: it stores a set of **unique** keys, and supports operations such as `add`, `remove`, etc. in expected $O(1)$ time. Read the Python documentation for more information.

(h) [3 points] Are tuples hashable in Python? How about arrays? Why or why not?

Hint: Python tries to maintain the following property: if `a == b`, then `hash(a) == hash(b)`.

(i) [4 points] Many security problems involve finding *hash collisions*, or multiple inputs that hash to the same value. For different values of d, find two 10-character strings whose Python hashes end in the same d bits (i.e. have the same remainder when divided by 2^d), and record how long it takes to find each collision.

As the value of d increases, how much harder is it to find a collision? At what value of d do you notice this takes too long? **Include a graph to support your answer.**

Note: For our purposes, "too long" means longer than a minute: unfortunately, we aren't elite hackers with millions of dollars of computing power.

(j) [10 points] **Extra Credit:** This problem was just a small look into how Python hash tables are implemented: this part is an opportunity to research further! For up to 10 extra credit points, is there anything else you can learn about how Python data structures are implemented? As in the previous parts, your writeup should include any relevant code, outputs, data, etc.

A few potentially interesting questions:

- So far, we've been focusing on Python dictionaries. Are there any significant differences when working with Python `list` or `set` objects?
- What happens during deletion?
- We haven't discussed the probe sequence, or the actual hash function Python uses. Anything you can say about either?
- ...anything else you find interesting!

Part B

Problem 3-5. [25 points] **Efficient Studying**

You are taking a very large number of classes, indexed 0 through $n-1$. For each class i you have estimated a maximum amount of time g_i that you can study for this class. You have also estimated for each class the benefit b_i that would result from studying for this maximum amount of time. Because these are just estimates, the values g_i and b_i will be integers less than 500.

Now, you have T time left before spring break, and you will spend it studying nonstop. Of course, you still will not be able to study the goal amount of time g_i for each of your classes. However, for each class you can study for any amount of time $t < g_i$, producing a benefit of $t \cdot b_i/g_i$.

Your goal is to find an efficient algorithm to determine the maximum (over all possible allocations of your time to your classes) total (additive) benefit you can obtain from studying.

You may only study for one class at once. You may not study a negative amount of time for any class. (You can assume that we won't give you any negative time values.)

In the file `study.py`, we have provided for your convenience a highly inefficient solution to this problem.

Submit your improved `study.py` to `alg.csail.mit.edu`. As usual, the autograder will run a more extensive suite of tests than `tests.py`, which should not be considered an accurate test of the efficiency of your algorithm. Your code will be run as `python3`. (You may submit to the autograder as many times as you want until the deadline; only the last submission to finish running before the deadline will count.)