# Question 1

1. **Proof that the Set is Universal**

The goal here is to prove that the given set is indeed universal. That is, for any distinct keys , the number of functions in for which and collide is . In other words, the number of functions in that set of functions for which is at most the cardinality of the set divided by . The example given proves this for the particular case of , but the goal is to show that it works in general, i.e., for any finite .

To start, utilize the following claim: given the universal set of hash functions, , if we pick uniformly at random, then for all distinct .

Proof:

In words, this says that the probability for a collision is at most , and this makes sense because since there are buckets and the functions are chosen uniformly at random, then each function has an equal chance of being chosen. This means that each individual hash function in is itself universal, again since they are chosen at random, and all have probability . Now that each hash function is proven to be universal, this leads to the proof that the entire set is also universal. If each function in has probability , then the probability for the entire set , given is the cardinality of , is:

The probability of the entire set is simply the linear addition of the probability of each hash function in . Thus, the set is universal with the probability of at most .

1. **Practicality of this Method for Generating Universal Sets**

Given this universal set , there is only one way to choose from it, which is uniformly at random. The way is generated, given is to simply adopt the set of all functions with domain and co-domain , where the domain pertains to the finite universe of keys, and the co-domain pertains to the number of buckets. The goal here is to describe the practicality of adopting such a method for generating this set as the universal set.

Based on this method, this is not a practical way for generating a universal set. Once the hash function is chosen from the universal set, there exists a need for encoding or representing this hash function in some way, for the purposes of storing it alongside the hash table and utilizing it in some useful manner. A natural way to encode or represent these functions is with a table. For example, from part (a), given a function , one could store it as a table, which contains, for each , whatever values from the function maps to. More specifically, given domain , where is the number of keys, and co-domain , the first column of the table could be the keys , and the second column could be what each is mapped to:

|  |  |
| --- | --- |
| **Keys** | **Values** |
|  |  |
|  |  |
|  |  |
|  |  |

The downfall of this method pertains to the size of this encoding for such a table. For each key, since there are buckets, then there are different ways in which each key can be mapped, since each key must correspond to at least one bucket. Furthermore, since there are keys, then the total size of this table is . This is a large amount of space to take up for such a task, especially considering if this were to get scaled up. Introducing a new key would add an entire set of mappings to the table, since it is possible for this new key to map to anything else already present in the table. Hence, this is not a practical way of generating a universal set, and the following paragraph describes a better method.

For a hash function that is chosen from the universal set that is proposed in the lectures, the data does not need to be stored as a table. This function, , given some prime such that , some key , , , and number of buckets , can be represented as:

Recall back to the tabular method, if one could control , then the hash function could have been constructed concisely without use of a table, however is strictly a finite universe of keys, nothing else. In the case of this modular-based function above, there is more freedom, in that can be modified to allow a more efficient encoding of the mapping. In the lecture, the domain for that example was the set of digit numbers to represent Student IDs as the keys. This allows the mapping to exist without the need of a table for keeping track of the key-value pairs, resulting in improved efficiency and reduced space usage. In this case where is not modifiable, there does not seem to be a compact and unambiguous encoding of the hash function that is chosen from that universal set. Therefore, the method of generation of universal sets given in this problem is not a practical method.

# Question 2

Example:

The items are but not necessarily sorted.

When we break up the array into pieces of , we get:

The median of the first piece of size is , the second is , and the third is .

The median of is . So, on this input array would return .

Another example:

Subsequences:

The medians are:

If (to find the smallest number), then the array would be split around and the algorithm recurses on the left portion. This portion is of size .

Clearly, , so the claim holds for this case.

Another example:

The medians are:

If , then the array would be split around and the algorithm recurses on the left portion. This portion is of size . Mistake? No:

Recall partitioning or splitting around the pivot :

Rearrange items in as follows:

* Move every item to the left of .
* Move every item to the right of .
* Thus, every item whose value is ends up where it would in a sorted permutation of .

So, if we split around as the pivot, we end up with the array as:

, where the first indices of the array contain the items from the set not necessarily in sorted order. Then, in index , you will have exactly … because the pivot ends up in the “right” spot. And the last indices will contain not necessarily in sorted order.

Using the algorithm on with (index of the value ), end up with:

And now, as , recurse on the part of the array to the left of , which is , with size which is

# Question 3

# Question 4

* The size of the input, , is the size of the graph:
* For the coding part, can use an array-based implementation of a priority queue

# Question 5

* If we have one more egg, we converge much faster, in the limit for the number of steps
* Worst case number of drops of an egg you need for a building of floors given eggs is . For all :

With we have:

Since we just have to do a linear search. With :

Since the best we can do is to drop the egg from floor, then do a linear search for the rest. Calculating the limit for these two cases:

# References

**There are no sources in the current document.**