15-451/651 Algorithms, Fall 2019

Homework #4
Due: October 21–23, 2019

This is an oral presentation assignment. There are three problems to be presented orally. You should work in groups of three. The sign-up sheet will be online soon (details on Piazza) and your group should sign up for a slot. Each person in the group must be able to present every problem. The TA/Professor will select who presents which problem. You are not required to hand anything in at your presentation, however having a written solution is often helpful for doing the presentation.

The bonus problem, B4, is a written problem, and can be turned into gradescope. The program is problem 5. It will be due Saturday October 26th at 11:59pm.

1. (25 pts) Cuckoo Hashing

In this problem we consider a hash method named after the Cuckoo bird, an open addressing method. In the simplest version we have a universe $U$ of keys and a subset $S$ of $n$ keys we wish to store is a hash table $A$ of size $M \geq n$. We have two hash functions $h_0$ and $h_1$. $h_i : U \rightarrow [0, M - 1]$.

Suppose we wish to hash $x \in U$ into the table $A$. We first check to see if either $h_0(x)$ or $h_1(x)$ are empty if so we store $x$ into one of these two locations.

1: function Cuckoo($k \in U, i \in \{0, 1\}, h_0, h_1$)
2: if $h_i(k)$ is empty then
3:   Value($h_i(k)$) $\leftarrow (k, i)$ return
4: else
5:   $(k', j) \leftarrow Value(h_i(k))$;
6:   Value($h_i(k)$) $\leftarrow (k, i)$; return Cuckoo($k', 1 - j$)
7: end if
8: end function

Observe that procedure CUCKOO could go into an infinite loop. In the case when CUCKOO halts we get a hash table that only requires two look ups per query. The goal of this problem is for you to understand cuckoo hashing in terms of bipartite matching.

(a) The code above is for historical reasons but it can be more easily seen as simply finding a matching in a bipartite graph. Describe a $d$-ary version of Cuckoo hashing by introducing a bipartite graph such that there exists a hashing using the $d$ hash functions if and only if there is a match of all the left nodes to right nodes. Describe the function CUCKOO as particular version of finding an augmenting path.

(b) Suppose we hope to hash $n$ elements into a table of size $2n$ using a 4-ary Cuckoo hashing. In this, and the following parts, our goal is to show that this will be possible with high probability.
We will assume the four hash functions are completely random. Let $X$ be a subset of $m$ keys we wish to hash and let $X'$ be $m$ locations which we hope to insert these keys $X$ using the four hash functions. Show that

$$\text{Prob}[\text{The images of the hash functions of } X \text{ is contained in } X'] \leq \left( \frac{m}{2n} \right)^4 m$$

(c) Show that the number of $X$ and $X'$ pairs we need to consider is bounded by

$$\binom{n}{m} \binom{2n}{m} \leq \left( \frac{ne}{m} \right)^m \left( \frac{2ne}{m} \right)^m \leq \left( \frac{ne}{m} \right)^m$$

(d) We next need to use the union bound to show that there will be a hashing with high probability. Finish the proof that with high probability the 4-ary Cuckoo hash with memory $2n$ will almost always be able to hash the keys. In particular, use parts 1b and 1c to show that with probability at least $1 - 1/n$, when the number of keys $n \geq 200$, can be hashed.

Hint: Show that for any fixed $m$ the failure probability will be at most $1/n^2$.

Hint: It may help to consider the cases $m \leq n/e$ and $m > n/e$ separately.

Hint: It may be useful that the function $\left( \frac{n}{m} \right)^m$ for $1 \leq m \leq n$ obtains a maximum at $m = n/e$ by taking it logarithm.

You may need the following identities:

$$\binom{n}{k}^k \leq \binom{n}{k} \leq \binom{ne}{k}^k$$

There is also a very famous theorem in matching theory called Hall’s theorem that may be useful.

**Theorem 1.** If $G = (A, B, E)$ is a bipartite graph then there is a match of all nodes in $A$ to those in $B$ if and only if for all $X \subseteq A$ we have that $|\text{neigh}(X)| \geq |X|$.
2. (25 pts) **How Universal?**

In this problem we will modify the matrix method to construct a hash family that you will analyze for $L$-universality. (For the definition of $L$-universal, see the end of section 3.2 of lecture 13.)

As in lecture, let the keys be $u$-bits long, and let the table size be $M = 2^m$. So an index is $m$-bits long. This time we pick two random $m \times u$ 0/1 matrices called $A_1$ and $A_2$. Our hash function is defined as follows:

$$h(x) = A_1x \oplus A_2 \bar{x}.$$  

The matrix-vector product uses modulo 2 arithmetic. The $\oplus$ indicates the exclusive or of two vectors of length $m$, and the symbol $\bar{x}$ indicates the bitwise complement of $x$. This defines a hash family $H$.

(a) Prove that $H$ is 2-Universal.

(b) Prove that $H$ is 3-Universal.

(c) Prove that $H$ is not 4-Universal.

3. (25 pts) **The Path Decomposition Theorem**

**Theorem.** Let $G = (V, E, s, t)$ be a flow graph with integer capacities and a maximum flow of $F$. Then there exists a sequence $S$ of $F$ simple paths in $G$ from $s$ to $t$ such that if an edge $e$ of $G$ occurs more than once among all the paths in $S$, then all usages of $e$ are in the same direction.

Prove this theorem by giving an algorithm that takes $G$ as input and produces a sequence $S$ of $F$ paths with the required properties.

4. (bonus) **Non-Attacking Rooks on a Restricted Chessboard**

Start with an $n \times n$ chessboard. The goal is to put as many non-attacking rooks on the board as possible. (Two rooks attack if they occupy the same row or column of the chessboard.) Of course it’s trivial to put $n$ non-attacking rooks on a diagonal.

However in this problem there is also a list of $k$ disjoint forbidden rectangles. Each one designates an axis-aligned rectangular region of the board on which you are not allowed to place any rooks.

Show how to set this up as a maximum flow problem with $|V| = O(n \log n)$ vertices and $|E| = O(k+n \log n)$ edges. The time to construct the flow graph is $O(|E|+|V|+k \log k)$. 

3
5. (25 pts) Palindromes with One Mistake

As you certainly know, a palindrome is a string that is the same when reversed. A palindrome with one mistake is a string with the property that you can change one character of it to make it a palindrome. An example is “dodad”.

The input to the program is a string of lower and upper case letters. The output is the longest (contiguous) substring of the input string that is a palindrome or a palindrome with one mistake (whichever is longer). If there are two that are equally long, output the leftmost one.

The input size is at most $10^6$ characters, and the time limit is 10 seconds.

For example, if the input is:

```
dogaaaaTaacat
```

then the output is:

```
aaaaTaaca
```

Second sample input:

```
dogaaaTaaacat
```

then the output is:

```
gaaaTaaac
```

Your algorithm should be guaranteed to give the right answer, even for inputs that an adversary chooses after looking at your code. So, for example, if your algorithm is randomized, it should be a Las Vegas algorithm. (That is, an algorithm which is guaranteed to give the right answer, but whose running time is a random variable.)

Include a comment in your submission that explains the algorithm that you used, along with a rough analysis of its running time (or expected running time, if your algorithm is randomized).