Advanced Algorithms and Data Structures
Assignment 3

Isabella Odorico	Caroline Kierkegaard	Mikkel Willén	Fie Hammer
jkd427	qlj556	bmq419	gsr530

January 3, 2024

Hashing

2.1

The truly independent hash function $h : U \to [m]$ is universal, since, if we have a value $h(x) \in [m]$, the chance of getting the same value $h(y) \in [m]$ uniformly at random, must be $\frac{1}{m}$, which is the definition of a universal hash function.

2.2

If a hash function $h : U \to [m]$ has collision probability 0, the size of m must be greater than or equal to the size of U. Thus, all the keys in the universe can be mapped to a distinct integer in m.

2.3

The identity function $f(x) = x$ is a universal hash function $[u] \to [m]$, since for $a \in u$ and $b \in u$ where $a \neq b$, the functions returns different hash values and since $u \leq m$ we have that $|L(f(x))| \leq |L(x)|$.

2.4

a.

We want to calculate the expected number of elements in $L[h(x)]$ if $x \in S$.

$$\begin{aligned}
\mathop{\mathbb{E}}_{h}[|L[h(x)]|] &= \mathop{\mathbb{E}}_{h}[|\{y \in S,\ x \in S \mid h(y) = h(x)\}|] \\
&= \mathop{\mathbb{E}}_{h}\left[\sum_{y \in S}[h(y) = h(x)]\right] \\
&= \sum_{y \in S} \mathop{\mathbb{E}}_{h}[[h(y) = h(x)]] \\
&= \sum_{y \in S \setminus \{x\}} \Pr_{h}[h(y) = h(x)] + 1 + \\
&\leq |S|\frac{1}{m} + 1 \leq \frac{n}{m} + 1 \leq 2
\end{aligned}$$

The expected number of elements in $L[h(x)]$ if $x \in S$ is less than or equal to 2.

b.

We calculate the bound we get, if h is only 2-approximately universal, assuming that x is not in S. If h is only 2-approximately universal, we have a different probability of collision. We have

$$\Pr_{h}[h(y) = h(x)] \leq \frac{2}{m}$$

and get the bound to be

$$\mathop{\mathbb{E}}_{h}[|L[h(x)]|] \leq |S|\frac{2}{m} \leq \frac{2n}{m} \leq 2$$

2.5

We find the probability of a false positive, when we search x.

$$\begin{aligned}
\Pr[\exists y \in S \mid h(y) = h(x) \wedge s(y) = s(x)] &= \Pr_{h}[\exists y \in S \mid h(y) = h(x)] \cdot \Pr_{s}[\exists y \in S \mid s(y) = s(x)] \\
&\leq \sum_{y \in S} \Pr_{h}[h(y) = h(x)] \cdot \sum_{y \in S} \Pr_{s}[s(y) = s(x)] \\
&\leq \frac{1}{n} \cdot \frac{1}{n^3} \\
&= \frac{1}{n^4}
\end{aligned}$$

2.6

a.

If $a = 0$, then the value of x is irrelevant for the hash function, since it is timed with 0. This means all hash values from $[p] \to [m]$ will be the same value.

b.

We want to prove that the given probability is always 2-approximately universal, that is for any distinct $x, y \in [p]$,

$$\Pr_{(a,b)\in[p]^2}[h_{a,b}(x) = h_{a,b}(y)] < \frac{2}{m}$$

We can slip the probability into two exclusive events. Either $a = 0$ or $a \neq 0$. The probability of $h_{a,b}(x) = h_{a,b}(y)$, when $a = 0$ is 1, since all values of b hash to the same value. The probability of $h_{a,b}(x) = h_{a,b}(y)$, when $a \neq 0$ is upper bounded by $\frac{1}{m}$, because in that case the hash function is universal and $x \neq y$.

$$\begin{aligned}
\Pr_{(a,b)\in[p]^2}[h_{a,b}(x) = h_{a,b}(y)] &= \Pr_{a=0,b\in p}[h_{a,b}(x) = h_{a,b}(y)] \cdot \Pr[a = 0] \\
&+ \Pr_{a\neq 0,b\in p}[h_{a,b}(x) = h_{a,b}(y)] \cdot \Pr[a \neq 0] \\
&\leq 1 \cdot \frac{1}{p} + \frac{1}{m} \cdot \left(1 - \frac{1}{p}\right) \\
&\leq 1 \cdot \frac{1}{m} + \frac{1}{m} \cdot \left(1 - \frac{1}{p}\right) \\
&< \frac{2}{m}
\end{aligned}$$

The 4th line follows from the 3rd, since $\left(1 - \frac{1}{p}\right)$ is very close to 1, but a little smaller, and therefore we know the 4th line is strictly smaller. Thus the probability is 2-approximately universal.

3.1

Generalize 2-independence. What is 3-independence? k-independence?
2-independence is a strong universal and concerns a pair of two events with distinct keys that are hashed to the same value. A random hashfunction is 2-independent (strong universal) if the probability of every pairwise event is:

$$Pr[h(x) = q \wedge h(y) = r] = Pr[h(x) = q] \cdot Pr[h(y) = r] = \frac{1}{m^2}$$

3-independence is whether 3 distinct keys are hashed to the same value. The probability is:

$$Pr[h(x) = q \wedge h(y) = r \wedge h(z) = s] = Pr[h(x) = q] \cdot Pr[h(y) = r] \cdot Pr[h(z) = s] = \frac{1}{m^3}$$

This shows us a pattern of how the independence and probability correlate, and here we can use k-independence to describe the probability of k distinct keys hashed to the same value, which can be described as $\frac{1}{m^k}$.

3.2

If h is c-approximately strongly universal, what is an upper bound on the pairwise event probability, $Pr[h(x) = q \wedge h(y) = r]$?
By using the fact the c-approximately strongly universal property, states that $Pr[h(x) = q] \leq \frac{c}{m}$, we can write:

$$\Pr[h(x) = q \mid h(y) = r] \leq \frac{c}{m} \wedge \Pr[h(y) = r] \leq \frac{c}{m}$$

This we can now substitute in to the conditional probability formula:

$$\Pr[h(x) = q \wedge h(y) = r] \leq \left(\frac{c}{m}\right) \cdot \left(\frac{c}{m}\right) = \frac{c^2}{m^2}$$

So the upper bound of the given pairwise event probability is $Pr[h(x) = q \wedge h(y) = r] \leq \frac{c^2}{m^2}$

3.3

Argue that if $h : U \to [m]$ is c-approximately strongly universal, then h is also c-approximately universal. (you must prove that the collision probability is at most $\frac{c}{m}$. It is not sufficient to prove $\frac{c^2}{m}$).
As stated we have to prove that the collision probability is at most $\frac{c}{m}$.
The collision probability is the sum of all hashed values of the product of the $Pr[h(x) \wedge h(y) = (q)]$ and so we can write it as such:

$$\sum_{q \in [m]} \Pr[h(x) = q] \cdot \Pr[h(y) = q]$$

As we established earlier, $Pr[h(x) = q] \leq \frac{c}{m}$, and so we can substitute this is the equation:

$$\sum_{q \in [m]} \frac{c}{m} \cdot \Pr[h(y) = q]$$

Now, we sum over all hash values and factor out $\frac{c}{m}$:

$$\frac{c}{m} \sum_{q \in [m]} Pr[h(y) = q]$$

As we're dealing with the probability for a sum of hash values, we know that the sum of a probability should always add up to 1. Therefore we can set our sum to equal 1:

$$\frac{c}{m} \cdot 1$$

Here we can see that the collision probability is at most $\frac{c}{m}$, which also shows that a c-approximately strong universal hash function is also c-approximately universal in terms of collision probability.

3.4

Following the proof of (10) from the lecture notes on hashing, we have a collision on $h_a(x) = h_a(y)$ if $a \cdot x$ and $a \cdot y = a \cdot x + a \cdot (y - x)$ are identical on bits $w - l, \ldots, w - 1$. It follows that these pairs of does not hash independently, since they hash to the same value, thus `Multiply-Shift` is not c-approximately strongly universal for any constant c.

3.5

Given $S_{h,t}(B)$ and $S_{h,t}(C)$, we estimate the size of the symmetric difference $(B \backslash C) \cup (C \backslash B)$. First, we formulate the problem.

$$\begin{aligned}
\mathop{\mathbb{E}}_{h}\left[S_{h,\,t}(B \backslash C) \cup S_{h,\,t}(C \backslash B)\right] &= \mathop{\mathbb{E}}_{h}\left[\sum_{x \in B \backslash C}[h(x) < t] \cup \sum_{y \in C \backslash B}[h(y) < t]\right] \\
&= \sum_{x \in B \backslash C} \sum_{y \in C \backslash B} \mathop{\mathbb{E}}_{h}[h(x) < t \cup h(y) < t] \\
&= \sum_{x \in B \backslash C} \sum_{y \in C \backslash B} \Pr_{h}[h(x) < t \cup h(y) < t] \\
&= \sum_{x \in B \backslash C} \Pr_{h}[h(x) < t] + \sum_{y \in C \backslash B} \Pr_{h}[h(y) < t] \\
&= \sum_{x \in B \backslash C} \frac{t}{m} + \sum_{y \in C \backslash B} \frac{t}{m} \\
&= |B \backslash C| \cdot \frac{t}{m} + |C \backslash B| \cdot \frac{t}{m}
\end{aligned}$$

3.6

We want to find an upper bound for the probability of $|X - \mu| \geq 10000$. From lemma 3.2 we know that

$$\Pr[|X - \mu| \geq q\sqrt{\mu}] \leq \frac{1}{q^2}$$

So we have

$$\begin{aligned}
\Pr[|X - \mu| \geq q\sqrt{\mu}] &\leq \frac{1}{q^2} \\
\Pr[|X - \mu| \geq q \cdot 1000] &\leq \frac{1}{q^2} \\
\Pr[|X - \mu| \geq 10000] &\leq \frac{1}{10^2} \leq \frac{1}{100}
\end{aligned}$$

The probability is upper bounded by $\frac{1}{100}$.

NP-Completeness

34.1-1

We want to show that the optimisation problem `LONGEST-PATH-LENGTH` can be solved in polynomial time if and only if the decision problem `LONGEST-PATH` is in the complexity class `P`.
For this optimisation problem, `LONGEST-PATH-LENGTH`, we have an undirected graph G and two vertices u and v, and define it, as finding the longest simple path between vertices u and v in G (the path with the most number of edges). So, `LONGEST-PATH-LENGTH` can be seen as a function which checks for all values $k \in \{1, ..., |E|\}$.
For the decision problem, `LONGEST-PATH` we have an undirected graph G, two vertices u and v, and an integer k ($k \geq 0$). We define this decision problem as checking if the longest path in G has k edges. We also assume this runs in polynomial time. So, `LONGEST-PATH` = {G, u, v, k} where $|E|$ is the number of edges in G.
This means that the decision problem algorithm is run with increasing values k until it returns true (this then shows that a path of lengths at least k exists), within the function of `LONGEST-PATH-LENGTH`. The function then terminates and returns the length k as shown in a pseudo code below:

```
LONGEST-PATH-LENGTH
    edges = |E|
    for (k = 1) to edges
        if LONGEST-PATH = True
            return k
    return 0
```

Since we already assumes that `LONGEST-PATH` runs in polynomial time, we can write `LONGEST-PATH`'s running time as $O(n^c)$ for a constant c. Looking at our pseudocode, we can then write the running time for `LONGEST-PATH-LENGTH` as $O(n \cdot n^c)$, which gives it the running time $O(n^{c+1})$.

34.1-5

We want to show that if an algorithm makes at most a constant number of calls to polynomial-time subroutines and performs an additional amount of work that also takes polynomial time, then it runs in polynomial time.
We assume the additional amount of works is performed on the output of the constant number of calls to polynomial time subroutines. This means we get get an output from the first part of the size n^{c_4} and thus we get

$$c_1 \cdot n^{c_2} + (n^{c_3})^{c_4} = c_1 \cdot n^{c_2} + n^{c_3 \cdot c_4} = O\left(n^{\max(c_2, c_3 \cdot c_4)}\right)$$

We also want to show that a polynomial number of calls to polynomial-time subroutines may result in an exponential-time algorithm. Lets say we have a routine that takes an input of n size and returns an output of double the size. This is our subroutine. We run the subroutine on the output of the previous subroutine n^{c_1} times, resulting in an algorithm with the run time

$$O\left(n^{c_1} + 2^{n^{c_2}}\right) = O\left(2^{n^{c_2}}\right)$$

which is exponential time.

34.2-5

If $k = |L|$ where L is the language, then the number of different possible combinations of the language for the size of a string n, will be

$$n^k$$

Then to check each possible combination of solutions to the NP problem will be

$$2^{n^k}$$

34.2-6

In order for the language to belong to NP, we must show that it can be verified by a polynomial-time algorithm.
If we know there is a path from u to v in the graph G, we can check all paths from u to another vertex, and then from that vertex and to yet another vertex. If we continue doing this, we get a running time of $O(n!)$ where n is the number of vertices in the graph G. This follows, since when we have a vertex u, we have $n-1$ vertices left, when we then move on we have $n-2$ vertices left and so on.

34.2-8

To show that `TAUTOLOGY` is in co-NP, we need to show that the complementary of `TAUTOLOGY` is in NP. The complementary of `TAUTOLOGY`, which we will call $\overline{\texttt{TAUTOLOGY}}$, will be, that there exists an input that evaluates to 0. To verify whether $\overline{\texttt{TAUTOLOGY}}$ gives a valid answer, we can input all the values in the boolean expression, and evaluate the result. This takes time equal to the number of inputs, and is therefore in polynomial time, which means $\overline{\texttt{TAUTOLOGY}} \in$ NP. It follows that `TAUTOLOGY` $\in$ co-NP by definition.

34.3-2

We want to show that the $\leq_P$ relation is a transitive relation on languages.
If we have

$$\forall\ L_1,\ L_2,\ L_3 \in L,\ \ ((L_1 \leq_P L_2) \wedge (L_2 \leq_P L_3)) \Rightarrow L_1 \leq_P L_3$$

where L is the set of all polynomial time reducible languages. If we can reduce $L_1 \leq_P L_2$ in polynomial time $O(P_1) = O(n^{c_1})$ for some constant c_1 and we can reduce $L_2 \leq_P L_3$ in polynomial time $O(P_2) = O(n^{c_2})$ for some constant c_2 with polynomial time computable functions, then we can reduce $L_1 \leq_P L_3$ in polynomial time $O(P_3) = O(P_1) + O(P_2) = O(n^{c_1} + n^{c_2})$, which shows $\leq_P$ is transitive.

34.3-3

We want to show that $L \leq_P \bar{L} \Leftrightarrow \bar{L} \leq_P L$. We have that

$$x \in L_1 \Leftrightarrow f(x) \in L_2 = x \notin L_1 \Leftrightarrow f(x) \notin L_2 \text{ and } x \in L \Leftrightarrow x \notin \bar{L}$$

therefore we have

$$x \in L \Leftrightarrow x \notin \bar{L} \Leftrightarrow f(x) \notin L \Leftrightarrow f(x) \in \bar{L}$$

and likewise

$$x \notin L \Leftrightarrow x \in \bar{L} \Leftrightarrow f(x) \in L \Leftrightarrow f(x) \notin \bar{L}$$

from this we have

$$L \leq_P \bar{L} \Leftrightarrow \bar{L} \leq_p L$$

as the functions, which is used for reduction, is bijective at the very least, and might be an involution.

Dispositions

Disposition of Hashing Isabella (jkd427)

- Introduction to Hashing
  - What is hashing (motivation)
  - Short example
- Different types of hashing
  - Universal
  - C-approximate
  - Strongly universal (k-independence)
- Hashing with chaining
- Show that Multiply-shift is 2-universal (with proof if time)

Disposition of NP Completeness Isabella (jkd427)

- Introduction to NP completeness
  - Language
  - Encoding
  - Decision Problems
  - Polynomial time (example)
- Different classes

  - P vs NP
  - Co-NP
- Explain verification algorithms
- NP-completeness
  - Reducibility
  - NP-hard
  - CIRCUIT-SAT example and proof (if time)

Hashing (Caroline)

- Talk about motivation for hashing
- Present different types of hashing: Universal, c-approximate universal and strongly universal
- Explain hashing with chaining
- Prove multiply-shift is 2-universal

NP Completeness (Caroline)

- Define language, encoding and decision problems
- Explain P vs NP (and co-NP)
- Explain verification algorithms
- Explain reducibility and NP-hard (NP-complete).
- Show CIRCUIT-SAT to be NP-complete.

Hashing (Mikkel)

- Introduction and motivation
- Types of hashing
  - Universal hashing
  - C-approximate universal hashing
  - Strong universal hashing
- Hashing with chaining
  - Example
  - Expected running time
- Proof Multiple-shift is 2-universal

NP Completeness (Mikkel)

- Introduction
  - P and NP
  - co-NP
  - Language and encodings
- Verification algorithms
- Reducibility and NP-hard (NP-complete)
- Show Circuit-sat to be NP-complete

Hashing (Fie)

- Why along with an example
- The different types:
  - Universal
  - C-approximate universal
  - Strongæy universaæ
- Hashing with chaining
- Proof that Multiply-shift is universal

NP Completeness (Fie)

- What is N completeness
  - Language, Encoding and Decision Problems
- P vs NP
- Co-NP
- Verification algorithms
- Reducibility and NP-Hard
- CIRCUIT-SAT - NP-Complete