AAD Aflevering 3

Andreas Hammer (qgk998), Anders Munkvad (tqs326), Johan Topp (fpg662)

December 2023

Hashing

2.1

We a truly random hash function h, which means that $h : U \to [m]$ assigns an independent uniformly random variable $h(x)$ to each key in x. That is for any hash $q \in [m]$ we have $\forall x \in U, \quad \Pr[h(x) = q] = \frac{1}{m}$. Meaning that for any two distinct keys $x, y \in U$ we have, $\Pr[h(x) = h(y)] = \frac{1}{m}$. Meaning a truly independent hash function is universal.

2.2

If a hash function $h : U \to [m]$ has collision probability 0, then for every element in U there must be a unique key in $[m]$. This means that m must satisfy $m \geq U$ for there to exist a hash function $h : U \to [m]$ where the collision probability is 0.

2.3

The identity function will simply hash any key to itself. Since $m \geq u$ this means we will have u unique hashes for every key in u. In this case any two distinct keys x, y will satisfy $h(x) = x$ and $h(y) = y$. And since we assume $x \neq y$ the collision probability is 0. Since $0 \leq \frac{1}{m}$, it is universal.

2.4

a

We wish to calculate $E_h[|L[h(x)]|]$. where $x \in S$. Since we know $x \in S$, the list must at least have length 1, since x is already there. Now we simply have to find the probability that another element of S hashed to $h(x)$. Meaning that the expected number of elements can be written as

$$E_h[|L[h(x)]|] = 1 + \sum_{y \in S/x} \Pr_h[h(y) = h(x)] \leq 1 + \frac{n-1}{m} \leq 2$$

b

In the case that h is only 2-approximately universal we have.

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

This means that the sum,

$$\sum_{y \in} \Pr_h[h(y) = h(x)] \leq 2 \cdot \frac{n}{m}$$

Giving us a new bound of

$$E_h[|L[h(x)]|] = \sum_{y \in} \Pr_h[h(y) = h(x)] \leq 2 \cdot \frac{n}{m} \leq 2$$

2.5

s and h are universal so the following holds for two distinct $x, y \in U$.

$$\Pr_s[s(x) = s(y)] \leq \frac{1}{n^3}$$

$$\Pr_h[h(x) = h(y)] \leq \frac{1}{n}$$

We know that if two events a, b are independent from one another we know that,

$$\Pr[a \wedge b] = \Pr[a] \cdot \Pr[b]$$

So that probability that a key $x \in U/S$ and $y \in S$ collide in both h and s is

$$\Pr_{s,h}[s(x) = s(y) \wedge h(x) = h(y)] = \Pr_h[h(x) = h(y)] \cdot \Pr_s[s(x) = s(y)] \leq \frac{1}{n} \cdot \frac{1}{n^3} = \frac{1}{n^4}$$

2.6

a

In the case that $a = 0$ is allowed, then there is a risk of selecting $(a, b) = (0, b)$ in which case the hashing matrix for that all x would evaluate to b this can be seen in the matrix table below, where a = 0, then the result will be $0 \cdot x + b = b \forall x$

x \b	0	1	2	...	k
0	0	1	2	..	k
1	0	1	2	..	k
2	0	1	2	..	k
3	0	1	2	..	k
..	0	1	2	..	k

so the odds of $x == b > \frac{1}{m}$ since all x hash to b
While if $a \neq 0$ then each x would hash to a different value, and the odds of $(x == b) = \frac{1}{m}$ since the table would look as the following in the case where $a = 1$ assuming $n + k < m$:

x \b	0	1	2	...	k
0	1	2	3	..	k+1
1	2	3	4	..	k+2
2	3	4	5	..	k+3
3	4	5	6	..	k+4
..	..	..	..	..	..
n	n	n + 1	n + 2	..	n+k

If $n + k \geq m$ then mod m would go into effect, wrapping around, from 0
It is important to note than once the b has been selected, only that column will be used, so the odds of getting 3 in this matrix is only 1 for $x \in [0, 3]$ even if it may seem that multiple different b values hash to 3, these are different hash functions in the same family of hash functions.

Taking this into consideration we can show by example that it may be possible for this hashing function not to be universal. If we randomly pick and a and b and just so happen to get $a = 0$ and $b = 4$. We can insert the values into our hashing function.

$$h_{0,4}(x) = ((0 \cdot x + 4) \mod p) \mod m = h_{0,4}(x) = (4 \mod p) \mod m$$

We can see that the hash value is independent from the x we give it meaning for all distinct $x, y \in [p],\ h(x)_{0,4} = h(y)_{0,4}$. If we compute the collision for this hashing function, no matter what kind of p or m we choose, we get,

$$\Pr_h[h(x)_{0,4} = h(y)_{0,4}] = 1$$

Meaning that for some choices of a and b the hashing function $h_{a,b}$ is not universal.

b

From (2)[1] and its proof, it is shown that when $a \in [P]_+$, then the probability of $Pr_{a\in[p]_+,b\in[p]}[h_{a,b}(x) = h_{a,b}(y)] < \frac{1}{m}$.
In the case where $a = 0$ then the probability $Pr_{a=0,b\in[p]}[h_{0,b}(x) = h_{0,b}(y)] = 1$.
The probability of being in the case where $a \neq 0$ is $\frac{p-1}{p}$ while the probability of being in the case where $a = 0$ is $\frac{1}{p}$. So the combined probability $Pr_{a\in[p],b\in[p]}[h_{a,b}(x) = h_{a,b}(y)] = \frac{1}{m}\frac{p-1}{p} + 1 \cdot \frac{1}{p}$.
Since $m < u < p$ then $\frac{1}{p} < \frac{1}{m}$ and $\frac{1}{m}\frac{p-1}{p} < \frac{1}{m}$ therefore $\frac{1}{m}\frac{p-1}{p} + 1 \cdot \frac{1}{p} < \frac{1}{m} + \frac{1}{m} = \frac{2}{m}$

3.1

We can extend 2-indenpedence to more events. For 3-independence we need the probability of 3 to be independent of each other, meaning for 3 distinct $x, y, z \in U$ and 3 hashes $q, w, r \in [m]$:

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

We can generalize this to k-independence, meaning that for k distinct keys $x_1, x_2, ..., x_k \in U$. And k not necessarily unique hashes $q_1, q_2, ..., q_k$. k-Independence must satisfy.

$$\Pr[h(x_1) = q_1 \wedge h(x_2) = q_2 \wedge \cdots \wedge h(x_k) = q_k] = \frac{1}{m^k}$$

3.2

$Pr[h(x) = q] \wedge Pr[h(y) = r] = Pr[h(x) = q] \cdot Pr[h(y) = r]$ since they are uniform with probability $\frac{c}{m}$ we get $Pr[h(x) = q] \cdot Pr[h(y) = r] = \frac{c}{m}\frac{c}{m} = \frac{c^2}{m^2}$

3.3

If h is c-approximately strongly universal, given distinct keys $x, y \in U$ and possibly non-distinct hash values $q, r \in [m]$ we have that,

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

For h to be c-approximately universal it must satisfy $\Pr[h(x) = h(y)] \leq \frac{c}{m}$.
Following the example set in Observation 3.1 we can write the following,

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

Meaning that h does satisfy the previously mentioned property and is thus c-approximately.

[1] page 4 of High Speed Hashing for Integers and Strings

3.4

No.
When looking at specific case of the function hash function $h(x) = \lfloor (ax \bmod 2^w)/2^{w-l} \rfloor$ where $a = 1, w = 64, l = 60$ this function goes from $[u] \to [m]$ where $[u]$ might be a 64 bit int and it hashes into $[m]$ of size 2^4.
Any int is hashed into itself, before **mod** and *division* is done.
Since $int_{64} \bmod 2^{64}$ will return itself, this part does nothing.
The division part though, removes the 60 least significant bits from the number. For all numbers $[0, 2^{60} - 1]$ this maps into bit value 0. While any int value $[2^{60} - 2^{64} - 1$ will map to $[0, 2^4 - 1$. The chance of hashing to 0 is therefore $((2^{60}) - 1)$ times higher, than any of the other hash values.

3.5

We can write the symmetric difference as $(C/B) \cup (B/C) = |C| + |B| - 2(B \cap C)$. Which we know can be written as.

$$|C| + |B| - 2(B \cap C) = S_{h,t}(B)\frac{m}{t} + S_{h,t}(C)\frac{m}{t} - 2(S_{h,t}(B)\frac{m}{t} \cap S_{h,t}(C)\frac{m}{t})$$

3.6

To give a bound on the probability we can write up the general formula from lemma 3.2 in the hashing notes. We can then insert our values for mu.

$$\Pr[|X - \mu| \geq q\sqrt{\mu}] \leq \frac{1}{q^2} = \Pr[|X - 1000000| \geq q \cdot 1000] \leq \frac{1}{q^2}$$

We can see that if we insert a value of $q = 10$, we get our bound.

$$\Pr[|X - \mu| \geq 10000] \leq \frac{1}{100}$$

NP-Completeness

34.1-1

For the optimization problem LONGEST-PATH-LENGTH the longest path between two vertexes can at most be of legnth $n = |E|$. This means we search for the solution using LONGEST-PATH as a subroutine to find the largest value k that satisfies this. We would need to run this at most $\log(n)$ times to find the solution. In this case given an *instance* of a solution to LONGEST-PATH-LENGTH, LONGEST-PATH can validate wether it is a solution or not. Meaning if LONGEST-PATH is not P, then LONGEST-PATH-LENGTH cannot be P either since the related decision problem to an optimization problem can be no harder. On the other hand if LONGEST-PATH is in P than LONGEST-PATH-LENGTH is also in P, since we showed that LONGEST-PATH-LENGTH can just be written as an instance of at most $lg(n)$ calls to a subroutine that it is P, which is still P.

34.1-5

Constant number

The first part of the exercise specifies an algorithm that does a constant number of calls to a polynomial algorithm and then also runs additional work in polymomial time. The algorithms runtime can be described as,

$$c \cdot n^a + n^b$$

To show this works in polynomial time we look at big-O notation. If we have a constant k and n_0 such that $0 \leq f(n) \leq k \cdot g(n), \forall n \geq n_0$, then $f(n) = O(g(n))$.
If we choose $g(n) = n^{\max(a,b)}$, then if $a > b$, then n^b is asymtotically smaller than n^a, and there exists a n_0 where it is made obsolete. This works the other way for $a < b$ too, and if $a = b$ then we can still reduce by a constant factor, meaning we can write the bound for large n_0, and $k \geq c$ as

$$c \cdot n^a + n^b \leq c \cdot n_0^{\max a,b} \leq k \cdot n_0^{\max a,b}$$

Meaning $g(n)$ satisfies the above mentioned requirements and the algorithm is $O(n^{\max(a,b)})$, making it polynomial time.

Polynomial number

If the polynomial function depends on c in the so that each call $(i = i{+}{+})$! then when the function is called a fixed number of times (c) then n_0 can just be chosen as c^c which is a fixed number, and for $n > c$ the function will be bounded by $O(n^c)$ but if the function is called n^c times, the function

34.2-5

A verification algorithm takes two inputs x, y one being the input and the other being a *certificate*. The algorithm can then verify the language if for x there exists y such that the algorithm returns 'yes'. If this verification runs in polynomial time, then the certificate y must be polynomial in size, and could be represented in binary. For any language in NP, there must exist an algorithm like this. We could then for a problem x always check every single possible certificate y, which must be bounded of size $O(n^k)$, bits. Checking every single possibility would mean all possible assignments of at most $O(n^k)$ bits, giving us that any language in NP can be solved in $2^{O(n^k)}$ time.

34.2-6

A certificate to the Hamiltonian path would be a list of vertexes. To verify this certificate we would need to check if the first vertex is u and the last vertex is v. Additionally we need to check if each vertex in G appears exactly once in the certificate, and finally we need to check for each pair (v_i, v_{i+1}) in the list of vertexes that there is an edge between them. All of these operations can be done in constant or polynomial time meaning that the language HAM-PATH belongs to NP.

34.2-8

A tautology is any logic ϕ in which any assignment of input variables $x_1, x_2, ..., x_k$. The language TAUTOLOGY (L) is then all logic statements which fulfill this requirement. The complement of a language is defined as $\bar{L} = \Sigma^* - L$, meaning the complement of TAUTOLOGY $(\bar{L})$ would be all logic statements where there exists an assigment of $x_1, x_2, ..., x_k$ where ϕ evaluates to 0. A certificate to the complement would be a set of assignments of $x_1, x_2, ..., x_k$, which we can validate whether or not evaluates to 0 in polynomial time, meaning if $\bar{L} \in NP$, then $L \in$ co-NP.

34.3-2

Polynomial-time reduceability written as "$L_1 \leq_{\mathrm{P}} L_2$" means that there exists a polynomial time function $f : \{0,1\}* \to \{0,1\}^*$ s.t $\forall x \in \{0,1\}^*$, $x \in L_1$ iff $f(x) \in L_2$.
This means for our case that if $L_1 \leq_{\mathrm{P}} L_2$ and $L_2 \leq_{\mathrm{P}} L_3$ then there exists a function $f(x)$ where $x \in L_1$ iff $f(x) \in L_2$ and a second function $g(x)$ where $x \in L_2$ iff $g(x) \in L_3$. We can see from this that if a value $f(x)$ is in L_2 then $g(f(x))$ has to be in L_3. We know that feeding the output of a polynomial-time algorithm into into another polynomial-time algorithm can be done

in polynomial time, so we can define a polynomial time algorithm $g(f(x)) : \{0,1\}* \rightarrow \{0,1\}^*$, that satisfies that $\forall x \in \{0,1\}*$, $x \in L_1$ iff $g(f(x)) \in L_3$, meaning that polynomial-time reduceability has the transitive property.

34.3-3

The definition of the complement is that $x \in \bar{L}$ iff $x \notin L$, and vice versa $x \in L$ iff $x \notin \bar{L}$. If $L \leq_{\mathrm{P}} \bar{L}$, it means there exists a function $f(x)$ s.t $x \in L$ iff $f(x) \in \bar{L}$. We can now show that this implies $f(x)$ is also a poly-time function that fulfils $\bar{L} \leq_{\mathrm{P}} L$.
We know that $x \in \bar{L}$ iff $x \notin L$, but if x is not in L it implies that $x \notin L$ iff $f(x) \notin \bar{L}$, but if $f(x)$ is not in the complement, then it must mean $f(x) \notin \bar{L}$ iff $f(x) \in L$. Since every step of this chain contains iff staments we can set the first step next to the last and we get $x \in \bar{L}$ iff $f(x) \in L$, which means $L \leq_{\mathrm{P}} \bar{L}$ implies $\bar{L} \leq_{\mathrm{P}} L$. We can do this the other way as well with the same logic to show that $L \leq_{\mathrm{P}} \bar{L}$ iff $\bar{L} \leq_{\mathrm{P}} L$.

1 Andreas Hammer

1.1 Hashing

- What is a hashing fucntion? (U, m)
- Truly random vs. universal vs. strongly universal
- hashing with chaining proof
- signatures (union bound)
- other hashing methods (multiply mod prime, multiply shift)

1.2 NP-completeness

- clear overview of definitions Σ, L, P, NP
- reductions and polynomial time reductions
- Extended NP definitions (NP-hard, NP-Complete, co-NP)
- Show Circut-SAT is NP-complete
- Show how reducing Circut-SAT to SAT implies NP-complete

2 Anders Munkvad

2.1 Hashing

- Why do we care about hashing?
- Definitions: Universal and strongly universal hashing.
- Example: Hash table with chaining.
- Showing that multiply-shift is 2-universal.

2.2 NP-completeness

- Outline: Problems that cannot be solved in polynomial time.
- Definition of NP and reduction
- NP-completeness.
- Example of NP-completeness

3 Johan Topp

3.1 Hashing

- Hashingfunction $[u] \rightarrow [m]$
- Random + independent
- Universal, c-universal
- Strongly universal + proof
- Multiply-shift, not not strongly universal for any constant c

3.2 NP-completeness

- Explain P and NP
- $P \subseteq NP$ (not proved, but assumed)
- Solving NP time vs verifying solution
- NP-hard, NP-complete
- Cook-Levin theorem
- Example of NP-complete: Clique and K-SAT
- Reduction from Clique K-sat