Hands-On 1: Algorithm Design 23/24

Filippo Boni Alessio Duè Giacomo Trapani

17 March 2024

1 Exercise 1

Describe and analyze how to transform the randomized Karp-Rabin string search algorithm from Monte Carlo to Las Vegas.

Solution

The main idea here is to consider the rolling hash technique, which in the standard implementation is a Monte Carlo algorithm (which may return false positives). Assuming that:

$$P \text{ is a pattern } P : [0, \dots, m-1]$$
$$T \text{ is a text } T : [0, \dots, n-1]$$

1.1 Algorithmic Implementation

A possible solution for the problem is to analyze algorithmically a version of the Karp-Rabin method that satisfies the given condition of no errors; here is the proposed approach:

Lemma 1. *Every positive integer a has at most $\log_2 x$ distinct prime divisors.*

Proof. By the *fundamental theorem of arithmetic*, we can write x as a product of primes $X = \prod_{i=1}^{k} p_i$; since $p_1 = 2$, then $x = \prod_{i=1}^{k} p_i \geq 2^k$. Therefore, it follows that $k \leq \log_2 x$. □

Lemma 2. *Let $\pi(x)$ be the cardinality of the set of prime numbers in the interval $[2; x]$. The value of $\pi(x)$ for $x \geq 2$ is approximately $\frac{x}{\log(x)}$.*

We use an operator $\text{hash}(s)$ that computes the hash for a string, and $U(h, x)$ that updates the rolling hash h with a new character x.

Here follows the pseudo-code for the Karp-Rabin Las-Vegas:

Algorithm 1 KARP-RABIN(T, P, hash, p)

```
m ← |P|
n ← |T|
target ← hash(P)
current_hash ← None
for i ∈ 1, ..., n − m do
    if current_hash ≠ None then
        current_hash ← U(current_hash, T[i + m])
    else
        current_hash ← hash(T[i : m])
    end if
    if current_hash = target then
        if Check_String_Equality(T[i : i + m], P) then
            Print("Match found")
            return (i, i + m)
        end if
    end if
end for
return None
```

1.2 Algorithm Analysis

1.2.1 False Matches

Theorem 1. *Assuming $m \geq 1$, $n \geq 2$, K constant, B a suitably large number in N, p a randomly chosen prime number in the interval $[2, B]$, Karp-Rabin reports a false match for a given position with a probability of at most $\frac{1}{K}$*

Proof. Let:

$$\text{target} = \text{hash}(P)$$

$$T_i = \text{hash}(T[i : i + m]);$$

Define:

- NoMatch to be the set of indices i for which $T_i \neq$ target, i.e., no match with target;
- $x = \prod_{i \in \text{NoMatch}}(|T_i - \text{target}|)$.

We know that x is a positive integer and since $|\text{NoMatch}| \leq n - m$ and $|T_i - \text{Target}| \leq 2^m$, we can say that $x \leq 2^{m(n-m)} \leq 2^{mn}$. By Lemma 1, we deduce that x has at most $m \cdot n$ prime divisors. Since we have a false match when $T_i \bmod p = \text{target} \bmod p$, then if there were a false match, p would divide x.

The following probability results are derived:

$$\Pr(\text{false match}) \leq \Pr(p \text{ divides } x) = \frac{\text{number of prime divisors of } x}{\pi(B)} \leq \frac{mn}{\pi(B)}$$

Choosing $B = K \cdot m \cdot n$, it follows that $\Pr(\text{false match}) = O\left(\frac{1}{K}\right)$. □

1.2.2 Complexity Analysis

Worst case we have a false match at every position: checking m characters n times $\Rightarrow O(m \cdot n)$.

Best case we have a true match in the first position: checking m characters once $\Rightarrow O(m)$.

Expected the probability of a false match at each position is $O(1/K)$, so the expected number of false matches is $O(n/K)$. The expected running time is $O(nm/K + n + m)$. We can choose K such that $m/K = c$ with c constant; for instance $K = m$. We then obtain $O(m + n)$.

The running time is not influenced by the value of K, except for the generation of the prime p, which is an additive logarithmic term in the asymptotic complexity and can thus be ignored.

2 Exercise 2

Show that $h_{ab}(x) = (ax + b \mod p) \mod m$ *is 2-independent.*

Solution

We call x_1, x_2 two values in U such that $x_1 \neq x_2$. We have to prove that

$$\Pr_{h \in H}[h(x_1) = y_1 \wedge h(x_2) = y_2] \leq \frac{1}{m^2}.$$

First, we consider the linear system

$$\begin{cases} z_1 = ax_1 + b \mod p \\ z_2 = ax_2 + b \mod p \end{cases}$$

Since p is prime, we know from Lagrange's theorem there is only one mapping from pairs (x_1, x_2) to (z_1, z_2), and we have $p \times (p-1)$ possible pairs of (z_1, z_2).

We notice the number of values in the same residual class modulo m has a lower bound in $\lfloor p/m \rfloor$ and an upper bound in $\lfloor p/m \rfloor + 1$.

We can now give the result

$$\frac{1}{p(p-1)} \times \left(\lfloor \frac{p}{m} \rfloor \right)^2 \leq \Pr_{h \in H}[h(x_1) = y_1 \wedge h(x_2) = y_2] \leq \frac{1}{p(p-1)} \times \left(\lfloor \frac{p}{m} \rfloor + 1 \right)^2$$

We can approximate $p(p-1)$ with p^2 and get the upper bound we have in the thesis. □

3 Exercise 3

Consider the deletion in cuckoo hashing: build an example so that the deletion does not produce a feasible graph, meaning that there is no insertion-only sequence that can lead to that graph. Show how to fix this by randomly choosing among h_1 *and* h_2 *when inserting.*

Solution

Consider the graph associated with the state of a Cuckoo hash table. It is specified by a set of nodes V. For each position i of the table T there is a node $i \in V$ if $T[i]$ is empty, or $(i, x) \in V$ if $T[i] = x$. We define the set of all valid sets of nodes as follows:

$$\begin{aligned}\mathbf{V} = \{V \subseteq \mathbb{Z}_m \cup (\mathbb{Z}_m \times X) \mid \forall i \in \mathbb{Z}_m.\\ (i \in V \vee \exists x \in X.\ (i, x) \in V)\\ \wedge (\forall x, y \in X.\ (i, x) \in V \Rightarrow\\ (i = h_1(x) \vee i = h_2(x))\\ \wedge\, i \notin V\\ \wedge\, ((i, y) \in V \Rightarrow x = y))\},\end{aligned}$$

where X is the set of values that can be put into the hash table.

The graph also has a set of undirected edges $E \subseteq \{\{v_1, v_2\} \mid v_1, v_2 \in V\}$. However, given the nodes V and the two hash functions h_1, h_2, E is uniquely determined as the set:

$$\begin{aligned}\text{edges(V)} =& \{\{(h_1(x), x), (h_2(x), y)\} \mid (h_1(x), x), (h_2(x), y) \in V\}\\ \cup& \{\{(h_1(x), y), (h_2(x), x)\} \mid (h_1(x), y), (h_2(x), x) \in V\}\\ \cup& \{\{h_1(x), (h_2(x), x)\} \mid h_1(x), (h_2(x), x) \in V\}\\ \cup& \{\{(h_1(x), x), h_2(x)\} \mid (h_1(x), x), h_2(x) \in V\}\end{aligned}$$

From now on we'll just deal with the nodes V, implicitly referring to the graph $(V, \text{edges}(V))$.

Let's define the functions $\text{ins}, \text{del}, \text{shift} : \mathbf{V} \times X \to \mathbf{V}$, which model the effect of inserting or deleting an element from the table.

$$\text{del}(V, x) = \begin{cases}(V \setminus \{(h_1(x), x)\}) \cup \{h_1(x)\} & (h_1(x), x) \in V\\ (V \setminus \{(h_2(x), x)\}) \cup \{h_2(x)\} & (h_2(x), x) \in V\end{cases}$$

$$\text{ins}(V, x) = \begin{cases}(V \setminus h_1(x)) \cup \{(h_1(x), x)\} & h_1(x) \in V\\ (\text{shift}(V, y) \setminus h_1(x)) \cup \{(h_1(x), x)\} & (h_1(x), y) \in V \wedge y \neq x\end{cases}$$

$$\text{shift}(V, x) = \begin{cases}(V \setminus \{(h_1(x), x), h_2(x)\}) \cup \{h_1(x), (h_2(x), x)\}\\ \qquad \text{if } (h_1(x), x) \in V \wedge h_2(x) \in V\\ (V \setminus \{(h_2(x), x), h_1(x)\}) \cup \{h_2(x), (h_1(x), x)\}\\ \qquad \text{if } (h_2(x), x) \in V \wedge h_1(x) \in V\\ (\text{shift}(V, y) \setminus \{(h_1(x), x), h_2(x)\}) \cup \{h_1(x), (h_2(x), x)\}\\ \qquad \text{if } (h_1(x), x) \in V \wedge (h_2(x), y) \in V\\ (\text{shift}(V, y) \setminus \{(h_2(x), x), h_1(x)\}) \cup \{h_2(x), (h_1(x), x)\}\\ \qquad \text{if } (h_2(x), x) \in V \wedge (h_1(x), y) \in V\end{cases}$$

shift(V, x) moves x from its current slot to its alternative slot (from $h_1(x)$ to $h_2(x)$ or vice versa). If the destination is already occupied by another element y, then y is shifted too. For simplicity, we assume no rehashing is necessary; shift(V, x) is not defined if there is a cycle (the recursion isn't well founded).

3.1 Problematic deletions

Let's consider a minimal example that shows how deletions can produce a graph which is not obtainable by insertions alone. Let $\mathcal{H} = \left\{h_{a,b} \mid a \in \mathbb{Z}_p^+, b \in \mathbb{Z}_m\right\}$, $h_{a,b}(x) = ((ax+b) \bmod p) \bmod m$, $p = 3$, $m = 2$, $h_1 = h_{1,0} \in \mathcal{H}$, $h_2 = h_{1,1} \in \mathcal{H}$. The hash table has 2 slots and contains value in $\mathbb{Z}_3$. Their hashes are:

x	0	1	2
h_1	0	1	0
h_2	1	0	0

The sequence of operations `insert 2, insert 0, delete 2` yields the table `[empty, 0]`. This is not obtainable via insertions alone, because if the first position is empty, `0` will be inserted there.

3.2 Solution with random hash function selection

Let's redefine ins as follows:

$$\text{ins}'(V, x) = \begin{cases} (V \setminus h(x)) \cup \{(h(x), x)\} & h(x) \in V \\ (\text{shift}(V, y) \setminus h(x)) \cup \{(h(x), x)\} & (h(x), y) \in V \wedge y \neq x \end{cases}$$

where h is choosen randomly between h_1 and h_2

This is of course not a function in the mathematical sense. We can more precisely treat it as a relation:

$$\text{ins}' = \left\{((V, x), V') \;\middle|\; \begin{array}{l} (h_1(x) \in V \wedge V' = (V \setminus h_1(x)) \cup \{(h_1(x), x)\}) \\ \vee\, ((h_1(x), y) \in V \wedge y \neq x \wedge V' = (\text{shift}(V, y) \setminus h_1(x)) \cup \{(h_1(x), x)\}) \end{array}\right\}$$

$$\cup \left\{((V, x), V') \;\middle|\; \begin{array}{l} (h_2(x) \in V \wedge V' = (V \setminus h_2(x)) \cup \{(h_2(x), x)\}) \\ \vee\, ((h_2(x), y) \in V \wedge y \neq x \wedge V' = (\text{shift}(V, y) \setminus h_2(x)) \cup \{(h_2(x), x)\}) \end{array}\right\}$$

$(V, x)\,\text{ins}'\,V'$ means that V' is one of the two possible outcomes of inserting x in the graph V.

Now we'll prove that:

1. given a valid graph, del always produces a valid graph (i.e. a graph in $\mathbf{V}$);
2. it is possible to produce any valid graph with just insertions with random hash function selection (ins′).

Together, these two propositions mean that any graph that can be produced by deletions can also be produced by insertions alone.

Proofs

1. $V \subseteq \mathbb{Z}_m \cup (\mathbb{Z}_m \times X)$ is in $\mathbf{V}$ if and only if, for all $i \in \mathbb{Z}_m$:

   (a) $i \in V \vee \exists x \in X.\ (i, x) \in V$

   (b) $\forall x \in X.\ (i, x) \in V \Rightarrow i = h_1(x) \vee i = h_2(x)$

   (c) $\forall x, y \in X.\ (i, x) \in V \Rightarrow i \notin V \wedge ((i, y) \in V \Rightarrow x = y))$

   If $V \in \mathbf{V}$, we show that for all $x \in X$, $\mathrm{del}(V, x)$ also has these properties, and thus is in $\mathbf{V}$.

   There are three cases to consider:

   $(h_1(x), x) \in V$**:** then $\mathrm{del}(V, x) = (V \setminus \{(h_1(x), x)\}) \cup \{h_1(x)\} = V'$. The three properties hold for V':

   (a) we remove $(h_1(x), x)$ from V, but we add $h_1(x)$ in its place, so it is still true that for all positions we have a node;

   (b) we don't add any node of the form (i, x), so this is trivially true;

   (c) similar to (a), we add $h_1(x)$ but remove $(h_1(x), x)$, which we know was in V, so we still have exactly one node corresponding to the position $h_1(x)$.

   $(h_2(x), x) \in V$**:** symmetric.

   $(h_1(x), x), (h_2(x), x) \notin V$**:** $\mathrm{del}(V, x) = V$.

2. The graph $V \in \mathbf{V}$ can be produced by ins' iff there exist $x_1, x_2, \ldots, x_k \in X$ and $V_1, \ldots, V_{k-1} \in \mathbf{V}$ such that:

$$\begin{gathered}(V_0, x_1)\,\mathrm{ins}'\,V_1 \\ (V_1, x_2)\,\mathrm{ins}'\,V_2 \\ \vdots \\ (V_{k-2}, x_{k-1})\,\mathrm{ins}'\,V_{k-1} \\ (V_{k-1}, x_k)\,\mathrm{ins}'\,V\end{gathered}$$

   where $V_0 = \mathbb{Z}_m$ is the graph corresponding to the empty hash table.

   We show that any valid graph $V_k = \{i_1, \ldots, i_{n-k}, (h(x_1), x_1), \ldots, (h(x_k), x_k)\} \in \mathbf{V}$ (where for each x_l, $h(x_l)$ is either $h_1(x_l)$ or $h_2(x_l)$) can be produced by ins', by induction on k:

   base case $k = 0$, trivial: $V = V_0$.

   inductive case we prove the property for $k + 1$ knowing that it holds for k. By inductive hypothesis,

$$\begin{gathered}(V_0, x_1)\,\mathrm{ins}'\,V_1 \\ (V_1, x_2)\,\mathrm{ins}'\,V_2 \\ \vdots \\ (V_{k-1}, x_k)\,\mathrm{ins}'\,V_k\end{gathered}$$

where $V_k = \{i_1, \ldots, i_{n-k}, (h(x_1), x_1), \ldots, (h(x_k), x_k)\}$.
It also holds that:

$$(V_k, x_{k+1}) \operatorname{ins}' V$$

since either $h_1(x_{k+1}) \in V_k$ or $h_2(x_{k+1}) \in V_k$ (otherwise V_k would have $k+1$ occupied slots, which by construction is impossible), and:

- if $h_1(x_{k+1}) \in V_k$ and $h_2(x_{k+1}) \notin V_k$, then

$$(V_k, x_{k+1}) \operatorname{ins}' \underbrace{(V_k \setminus h_1(x_{k+1})) \cup \{(h_1(x_{k+1}), x_{k+1})\}}_{=V_{k+1}}$$

- if $h_2(x_{k+1}) \in V_k$ and $h_1(x_{k+1}) \notin V_k$, then

$$(V_k, x_{k+1}) \operatorname{ins}' \underbrace{(V_k \setminus h_2(x_{k+1})) \cup \{(h_2(x_{k+1}), x_{k+1})\}}_{=V_{k+1}}$$

- if $h_1(x_{k+1}) \in V_k$ and $h_2(x_{k+1}) \in V_k$, then

$$(V_k, x_{k+1}) \operatorname{ins}' \underbrace{(V_k \setminus h_1(x_{k+1})) \cup \{(h_1(x_{k+1}), x_{k+1})\}}_{=V'}$$

$$(V_k, x_{k+1}) \operatorname{ins}' \underbrace{(V_k \setminus h_2(x_{k+1})) \cup \{(h_2(x_{k+1}), x_{k+1})\}}_{=V''}$$

  and $V_{k+1} \in \{V', V''\}$.

We conclude that

$$\begin{gathered}
(V_0, x_1) \operatorname{ins}' V_1 \\
(V_1, x_2) \operatorname{ins}' V_2 \\
\vdots \\
(V_{k-1}, x_k) \operatorname{ins}' V_k \\
(V_k, x_{k+1}) \operatorname{ins}' V_{k+1}
\end{gathered}$$

is a sequence of insertions that can (depending on the random choice of hash functions) generate V_{k+1}.