Aufgabe3: LaTeX-Dokument

Team: ??? / Name
Team-ID: ???

2. April 2022

Inhaltsverzeichnis

1 Lösungsidee

1.1 Problem a)

Gegeben sind N 128-Bit Zahlen $\{a_0, \ldots, a_{N-1}\} = Z$, die gegebenen Karten. (Indizes sind 0-basiert)
Gesucht sind k Indizes I, sodass $a_{i_0} \oplus \cdots \oplus a_{i_{k-2}} = a_{i_{k-1}}$ für $i_j \in I$, die Indizes der Karten, die nicht von den Freunden Zaras hinzugefügt wurden. (k hier ist um eins größer als in der Aufgabenstellung!)
Da $x \oplus x = 0$ und $\oplus$ assoziativ und kommutativ ist gilt:

$$a_{i_0} \oplus \cdots \oplus a_{i_{k-2}} = a_{i_{k-1}} \iff a_{i_0} \oplus \cdots \oplus a_{i_k-1} = 0 \tag{1}$$

Das Verfahren zum Finden der Menge I wird abgesehen vom nächsten Block im Rest der Dokumentation erklärt.

1.2 Problem b)

Hat man einmal die Menge I und damit die Schlüssel sowie die Sicherungskarte gefunden, stellt sich die Frage, wie Zara ein Haus ohne mehr als zwei Fehlversuche zu benötigen aufsperren kann.
Dies ist möglich, da die Codewörter aufsteigend sortiert sind. Das einzige Problem ist, dass die Sicherungskarte aufgrund von (1) nicht von den Schlüsseln unterscheidbar ist. Um ein Haus i aufzusperren muss Zara zuerst die i-te der sortierten Karten verwenden. Falls diese nicht passt weiß sie, dass die Sicherungs-Karte im Bereich [0;i) liegt und deshalb alle Indizes ab der Sicherungskarte um eins größer sind als sie seien sollten. Die gesuchte Karte ist dann die (i+1)-te

1.3 Generelle Definitionen

Es sind die 128-Bit Zahlen $a_0, \ldots, a_{N-1}$ gegeben.

$$Z := \text{Indizes in } [0; N)$$

$$\text{Für eine Indexmenge z: } xor(z) := \bigoplus_{x \in z} a_x$$

1.4 Naive Ansätze

Das Problem besitzt eine gewisse Ähnlichkeit zum Subset sum Problem für eine feste Anzahl an Summanden, einziger Unterschied ist, dass bei diesem Problem $\oplus$ statt + verwendet wird. Eine DP-Lösung, wie sie beim Subset sum Problem für kleine Summen möglich wäre kann hier nicht verwendet werden, da die Zahlen im Bereich $[0, 2^{128})$ liegen.
Ein anderer möglicher Ansatz wäre alle k-Kombinationen an Indizes in [0;N) auszuprobieren, doch für N=111 und k=11 gibt es $\binom{n}{k} \approx 4.7 \cdot 10^{14}$ viele Möglichkeiten. Diese alle einzeln auszuprobieren würde zu lange dauern.

1.5 Effizientere Durchquerung des Suchraums durch Hashtable Strukur

Die gegebenen Zahlen werden in zwei Hälften L und R geteilt. Angenommen folgende Information ist bekannt: l Indizes liegen in L und r Indizes liegen in R Eine l-Kombination an Indizes K_1 aus der ersten Hälfte ist dann teil der Lösungsmenge, wenn eine korrespondierende r-Kombination an Indizes K_2 aus der zweiten Hälfte existiert, so dass: (Zielwert?)

$$xor(K_1) \oplus xor(K_2) = 0$$
$$\implies xor(K_1) = xor(K_2)$$

Anstatt für alle $\binom{N/2}{l}$ möglichen K_1 jeweils alle $\binom{N/2}{r}$ möglichen K_2 auszuprobieren
(Laufzeit: $O(\binom{N/2}{l} \cdot \binom{N/2}{r})$),
können alle $\binom{N/2}{r}$ möglichen K_2 in einer Hashtable gespeichert werden, wo sie mit $xor(K_2)$ als Schlüssel in konstanter Zeit gefunden werden können (Laufzeit: $O(\binom{N/2}{l} + \binom{N/2}{r})$.
Für r=5 müssen nur $\binom{55}{5} \approx 3.5 \cdot 10^6$ viele Werte gespeichert werden. (Es sei an dieser Stelle angemerkt, dass das Durchsuchen von $\binom{n}{k}$ Kombinationen besser durch $O(\sum_{j=1}^{k} \binom{n}{j})$!!!!(Nur für kleine k) beschrieben wird. Da die Differenz jedoch relevant zur Schätzung der Laufzeit ist wird sie ignoriert.)
Das Problem ist nun aber, dass l und r nicht bekannt sind. Zur Lösung gibt es zwei Ansätze

1.6 Ansatz 1: Alle möglichen Paare l, r ausprobieren

Da es für k=11 nur 11 mögliche Paare (l, r) gibt wäre es denkbar all diese Paare auszuprobieren. Eine Hürde stellen die Extremfälle dar, in denen zum Beispiel l=0 ist. Dann gibt es nämlich $\binom{55}{11} \approx 1.2 \cdot 10^{11}$ Kombinationen rechts. Zwei Optimierungen werden benötigt:

1. Es sollen wenn überhaupt immer die Kombinationen der Hälfte mit weniger Kombinationen in der Hashtable gespeichert werden. So wird der benötigte Speicher reduziert, die Laufzeit bleibt gleich.

2. Falls die Anzahl an Kombinationen zu groß ist um komplett durchgegangen zu werden, soll der Suchalgorithmus für jede Kombination der geringeren Hälfte rekursiv auf die größere Hälfte angewandt werden.
   Im Falle l=0 würde der Vorgang also einmal rekursiv auf die rechte Hälfte angewandt werden.

Der gerade genannte Ansatz ist in der Praxis mit bestimmten Optimierungen schnell genug um das Beispiel mit n=111 und k=11 in ca. 13 Minuten zu lösen. Jedoch ist er zu langsam für die größeren Beispiele, schwer zu parallelisieren und unschön.

1.7 Ansatz 2: Verschieben der Hälften

Es sind die Paare (l, r) am aufwendigsten, bei denen l und r weit auseinander liegen:

$$\frac{\binom{56}{0}+\binom{55}{11}}{\binom{56}{5}+\binom{55}{5}} \approx 3.3 \cdot 10^3$$

Der folgend beschriebene Algorithmus iteriert über verschiedene Zusammensetzung der Hälften und belässt dabei die l r Verteilung gleich, so dass l und r so nah aneinander wie möglich sind. Es werden maximal $\lfloor \frac{N}{2} \rfloor + 1$ Iterationen benötigt und diese werden von 0 an in einer Schritten gezählt. Wie werden die Hälften zusammengesetzt und ist garantiert, dass alle Kombinationen betrachtet werden?
Die Hälften kann man sich als zusammenhängende Bereiche vorstellen, die bei jeder Iteration verschoben werden. Beispiel für n=8:

- s=0: ****—-
- s=1 -****—
- s=2 –****–
- s=3 —****-
- s=4 —-****

Allgemein entsprechen bei der Iteration s:
Die erste Hälfte $L_s := [s; \lceil N/2 \rceil + s)$
Die Zweite Hälfte $R_s := Z \backslash L_s$
Die Zweite Hälfte
$[(\lceil N/2 \rceil + s) \mod N; (N+s) \mod N) := [0; N+s \mod N) \cup [(\lceil N/2 \rceil + s \mod (N+1); N)$
Bild?
Hilfssatz: Für jede k-Kombination aus Z existiert mindestens ein $0 <= s <= \lceil \frac{n}{2} \rceil$ **, so dass** $\lceil \frac{k}{2} \rceil$ **der Elemente der k-Kombination in** L_s **sind**(und $\lfloor \frac{k}{2} \rfloor$ in R_s.
Aus dem Hilfssatz folgt die Korrektheit des Verfahrens, da dann jede k-Kombination bei mindestens einer Iteration in Betracht gezogen wird. Die Laufzeit des zweiten Verfahrens ist ca. $O(N * \binom{N/2}{k/2})$

1.7.1 Beweis des Hilfssatzes

Um denn Satz zu beweisen, reicht es zu zeigen, dass für eine beliebige k-Kombination C ein s gefunden werden kann, dass die erwartete Bedingung erfüllt.
x_s sei die Anzahl der Kombinationselemente in der ersten Hälfte: $x_s :=\mid K \cap L_s \mid$
y_s sei die Anzahl der Kombinationselemente in der zweiten Hälfte $y_s = k - x_s$
Zentral für den Beweis ist folgende Beobachtung über die Veränderung von x_s bei Änderung von s:

$$\mid x_{s+1} - x_s \mid = \begin{cases} 1 \\ 0 \end{cases} \tag{2}$$

Dies ergibt sich daraus, dass bei einer "Verschiebung"von L_s maximal ein Element entfernt werden muss und maximal eines hinzugefügt werden muss. Um den Satz zu beweisen muss nun gezeigt werden, dass $\lceil k/2 \rceil$ zwischen inklusiv x_0 und $x_{\lfloor n/2 \rfloor}$ liegt.
$u := \begin{cases} 1, \text{falls n ungerade ist und für das mittlere immer in } L_s \text{ enthaltene Element gilt} \lfloor n/2 \rfloor \in C \\ 0, \text{in allen anderen Fällen} \end{cases}$
$(x_0 = a + u)$ wird nach $\lfloor n/2 \rfloor + 1$ Iterationen zu $(x_{\lfloor n/2 \rfloor} = b + u)$
$a + u + b = k$
$\lceil k/2 \rceil$ zwischen inklusiv x_0 und $x_{\lfloor n/2 \rfloor}$ entspricht also
$\lceil \frac{a+u+b}{2} \rceil$ zwischen inklusiv $a + u$ und $b + u$ Es wird zwischen u=0 und u=1 unterschieden:
Im ersten Fall gilt: $\lceil \frac{a+b}{2} \rceil$ zwischen inklusiv a und b, da der Durchschnitt immer zwischen seinen zwei Teilen liegt.
Im zweiten Fall muss zwischen $(a + b) \mid 2$ und $(a + b) \nmid 2$ unterschieden werden.
Falls $(a + b) \mid 2$ gilt $\lceil \frac{a+b+1}{2} \rceil = \frac{a+b}{2} + 1$.
Das $\frac{a+b}{2} + 1$ zwischen a+1 und b+1 wird bereits im ersten Fall gezeigt. Falls $(a + b) \nmid 2$ wird zuerst angenommen, dass $a <= b$. Was folgt gilt auch für $b > a$, da a und b umtauschbar sind.

$(\lceil \frac{a+b+1}{2} \rceil = \frac{a+b}{2} + \frac{1}{2}$
$\frac{a+b}{2} + \frac{1}{2}$ ist kleinergleich b+1, da $a < b$.
a+1 ist kleinergleich $\frac{a+b}{2} + \frac{1}{2}$, da der einzige Fall in dem das nicht so wäre a=b fordert und dies $(a+b) \nmid 2$ wiederspricht.

1.8 Das komplette Verfahren

Der zweite Ansatz kann gut parallelisiert werden und ist somit in der Lage alle Beispiel bis auf 4 einigermaßen schnell zu lösen. Um alle Beispiele zu lösen bietet sich eine Kombination beider Ansätze an, die es erlaubt mehr Arbeit zu parallelisieren. Ein Flaschenhals bei der Parallelisation, wird sich in der Umsetzung herausstellen, ist Ram. Um alle 8 Kerne eines CPUs auszunutzen muss die Fragestellung erst mithilfe des 1. Ansatzes in kleinere Probleme geteilt werden, welche dann mit dem 2. Ansatz effizient parallel gelöst werden können. Mit diesem kombinierten Ansatz kann auch das schwierigste Beispiel in weniger als einer halben Stunde gelöst werden.

2 Umsetzung

Für die Umsetzung wurde Rust verwendet

3 Erweiterungen

4 Beispiele

5 Quellcode