1 Wprowadzenie

Będziemy rozwiązywać problem słownika. Dany jest zbiór elementów S, który jest podzbiorem uniwersum $U = \{1, \ldots, |U|\}$. Nasze zadanie to implementacja następujących operacji:

1. Lookup(x) – czy $x \in S$?
2. Insert(x) – $S := S \cup \{x\}$
3. Remove(x) – $S := S \setminus \{x\}$

Liczba elementów, które będziemy chcieli umieścić w słowniku, będzie ograniczona przez n.

2 Haszowanie przez łańcuchowanie

Weźmy tablicę $T[1 \ldots m]$ oraz funkcję $h : U \mapsto \{1, \ldots, m\}$. Zwykle $m = 2n$ lub $m = n$. Dla każdego j pozycja $T[j]$ zawiera *listę* takich elementów $x \in S$, że $h(x) = j$. Teraz możemy zaimplementować operacje *Lookup*, *Insert* i *Remove* na słowniku – dla danego x wykonujemy odpowiednią operację na liście $T[h(x)]$.

Pozostaje tak wybrać funkcję h, aby wszystkie operacje wykonywały się szybko.

Uwaga 2.1. Każda funkcja $h : U \mapsto \{1, \ldots, m\}$ jest *zła*, tzn. złośliwy przeciwnik, jeśli będzie znał h, może tak dobierać elementy wsadzane do słownika S, że $h(x) = h(y)$ dla każdych $x, y \in S$.

Z drugiej strony, funkcje h „jednostajne", tzn. $\forall_{u_1,u_2} |h^{-1}(u_1)| \approx |h^{-1}(u_2)|$, działają dobrze dla losowych danych – taka jednostajna funkcja to na przykład $h(x) = x \mod m$. Jednak u nas dane nie są losowe.

Pomysłem na poradzenie sobie z tym problemem będzie losowanie funkcji haszującej z pewnego zbioru funkcji – w ten sposób przeciwnik będzie zmuszony się zabezpieczyć przed całym zbiorem funkcji, a to nie będzie możliwe.

Przykład 2.2. Załóżmy, że h jest losowana jednostajnie ze zbioru $\{1, \ldots, m\}^U$. Odpowiada to wylosowaniu niezależnie dla każdego $x \in U$ wartości $h(x) \in \{1, \ldots, m\}$. Zauważmy, że wówczas

$$\mathbb{P}\left[h(x) = h(y)\right] = \begin{cases} 1 & \text{, gdy } x = y \\ \frac{1}{m} & \text{, gdy } x \neq y. \end{cases}$$

Stąd, średni czas wyszukiwania jest równy:

$$\mathbb{E}|h(x)| = \mathbb{E}\left[\sum_{y \in S} 1_{h(x)=h(y)}\right] = \sum_{y \in S} \mathbb{P}\left[h(x) = h(y)\right] = \begin{cases} \frac{n}{m} & \text{, gdy } x \notin S \\ 1 + \frac{n-1}{m} & \text{, gdy } x \in S. \end{cases},$$

Dla $m = \Omega(n)$ to wyrażenie jest $O(1)$.

Powstaje jednak pytanie jak szybko wylosować taką funkcję oraz ile pamięci potrzeba, aby ją reprezentować. Pierwsze, co przychodzi do głowy to reprezentowanie h jako tablicy $T[1 \dots |U|]$ liczb od 1 do m. Jeśli jednak mamy do dyspozycji tyle pamięci (i czasu na inicjalizację) to nie musimy używać haszowania – słownik możemy zaimplementować jako tablicę bitową rozmiaru $|U|$. Z drugiej strony, jeśli naprawdę uprzemy się, żeby h losować ze zbioru *wszystkich* funkcji, lepsze rozwiązanie nie istnieje: do przechowywania h potrzebujemy co najmniej $|U| \lg m$ bitów (za pomocą mniej niż $|U| \lg m$ bitów możemy reprezentować mniej niż $2^{|U| \lg m} = m^{|U|}$ funkcji, czyli nie wszystkie).

Zauważmy, że w powyższym przykładzie nie potrzebowaliśmy pełnej losowości, a jedynie $\mathbb{P}[h(x) = h(y)] \leqslant \frac{1}{m}$ dla $x \neq y$. Wystarczyłaby też niezależność parami i jednostajność zmiennych losowych $h(x)$ dla $x \in U$:

$$\begin{aligned}\mathbb{P}[h(x) = h(y)] &= \sum_{i \in \{1,\dots,m\}} \mathbb{P}[h(x) = i \wedge h(y) = i] \\ &= \sum_{i \in \{1,\dots,m\}} \mathbb{P}[h(x) = i] \cdot \mathbb{P}[h(y) = i] = m \cdot \frac{1}{m} \cdot \frac{1}{m} = m.\end{aligned}$$

2.1 Rodziny (α, k)-uniwersalne i k-niezależne

Definicja 2.3. Niech $\mathbb{H} \subseteq \{1, \dots, m\}^{|U|}$. Powiemy, że $\mathbb{H}$ jest rodziną (α, k)-*uniwersalną*, gdy jeśli wybraliśmy *losowo* $h \in \mathbb{H}$ (tzn. jednostajnie), to dla dowolnych parami różnych $x_1, x_2, \dots, x_k \in U$ zachodzi

$$\mathbb{P}[h(x_1) = h(x_2) = \dots = h(x_k)] \leqslant \frac{\alpha}{m^{k-1}}.$$

Jeśli $\alpha = 1$, to rodzina jest po prostu k-uniwersalna.

Definicja 2.4. Niech $\mathbb{H} \subseteq \{1, \dots, m\}^{|U|}$. Powiemy, że $\mathbb{H}$ jest rodziną *silnie k-uniwersalną* (lub k-niezależną), gdy jeśli wybraliśmy losowo $h \in \mathbb{H}$, to dla dowolnych parami różnych $x_1, x_2, \dots, x_k \in U$ i dowolnych $y_1, y_2, \dots, y_k \in \{0, \dots, m-1\}$ zachodzi

$$\mathbb{P}[h(x_1) = y_1 \wedge h(x_2) = y_2 \wedge \dots \wedge h(x_k) = y_k] = \frac{1}{m^k}.$$

Równoważnie:

- dla dowolnych parami różnych $x_1, \dots, x_k \in U$ zmienne losowe $h(x_1), \dots, h(x_k)$ są k-niezależne
- dla dowolnego $x \in U$ zmienna losowa $h(x)$ ma rozkład jednostajny, tzn. $\forall_{i \in \{1,\dots,m\}} \mathbb{P}[h(x) = i] = \frac{1}{m}$

Zauważmy, że silna uniwersalność implikuje zwykłą uniwersalność.

Z przykładu 2.2 wynika, że dla 2-uniwersalnej rodziny funkcji haszujących oczekiwany czas operacji *Insert*, *Lookup* i *Delete* jest stały. A jaki jest pesymistyczny czas?

$$\begin{aligned}&\mathbb{E}\left[\text{całkowita liczba kolizji}\right]\\ &= \mathbb{E}\left[\sum_{x,y\in S} 1_{h(x)=h(y)}\right] = \sum_{x,y\in S}\mathbb{P}\left[h(x)=h(y)\right] \leqslant \binom{n}{2}\frac{1}{m} < \frac{n^2}{2m}\end{aligned}$$

Z nierówności Markowa

$$\mathbb{P}\left[\text{całkowita liczba kolizji} \geqslant \frac{n^2}{m}\right] < \frac{1}{2}. \tag{1}$$

Ustalmy pozycję $i \in \{1,\ldots,m\}$ w tablicy, wtedy

$$\mathbb{P}\left[\text{liczba kolizji w } T[i] \geqslant \frac{n^2}{m}\right] < \frac{1}{2}.$$

Liczba kolizji na liście $T[i]$ jest równa $\binom{|T[i]|}{2}$, a zatem

$$\mathbb{P}\left[|T[i]| - 1 > \sqrt{\frac{2n^2}{m}}\right] < \frac{1}{2}.$$

Tak więc dla $n = \Theta(m)$ z prawdopodobieństwem co najmniej $\frac{1}{2}$ pesymistyczny czas wszystkich operacji wynosi $O(\sqrt{n})$. Trzeba przyznać, że to stwierdzenie niewiele nam mówi. Jeśli jednak weźmiemy $m = n^2$, to z nierówności (1) dostajemy, że z prawdopodobieństwem co najmniej $\frac{1}{2}$ nie będzie żadnych kolizji. Ten fakt okaże się kluczowy w kolejnym rozdziale.

2.2 Haszowanie doskonałe (Fredman, Komlós, Szemerédi)

Teraz będziemy rozważać tylko *statyczny* słownik, tzn. najpierw wrzucamy wszystkie n elementów, a potem wykonujemy tylko *Lookup*. Dla danego zbioru $S \subseteq U$ szybko (liniowo) zbudujemy strukturę danych, dzięki której *Lookup* będzie zajmował czas stały.

Faza I Wybrieramy losową funkcję haszującą $h : U \mapsto \{1,\ldots,n\}$ z pewnej 2-uniwersalnej rodziny funkcji haszujących (w punkcie 3.1 podamy przykład takiej rodziny). Na podstawie rozważań z poprzedniej sekcji, a w szczególności z (1) dla $m = n$, mamy wtedy

$$\mathbb{P}\left[\text{całkowita liczba kolizji w } S \geqslant n\right] \leqslant \frac{1}{2}.$$

Powtarzamy losowanie funkcji haszującej tak długo, aż liczba kolizji jest nie większa od n. Rozkład liczby powtórzeń jest zmajoryzowany przez rozkład geometryczny, więc oczekiwana liczba powtórzeń jest nie większa niż 2, a zatem oczekiwany czas pierwszej fazy jest $O(n)$.

Faza II Niech $S_i = \{x \in S |\ h(x) = i\}$. Zbiory S_i stanowią podział S, tzn. $S = S_1 \cup S_2 \cup \ldots \cup S_n$ i $S_i \cap S_j = \emptyset$ dla $i \neq j$. Dla każdego $i = 1, \ldots, n$ mamy tablicę drugiego poziomu $T_i[1 \ldots |S_i|^2]$. Tablica pierwszego poziomu $T[i]$ przechowuje adres tablicy drugiego poziomu T_i.

Teraz dla każdego $i = 1, \ldots, n\}$ chcemy przypisać elementy S_i do komórek w T_i tak, aby *nie było żadnych kolizji*. W tym celu, dla każdego $i = 1, \ldots, n\}$ losujemy dla $h_i : U \to \{1, \ldots, |S_i|^2\}$ z rodziny 2-uniwersjalnej tak długo, aż nie ma kolizji. Znowu, wstawiając do (1) $n = |S_i|$ oraz $m = |S_i|^2]$ otrzymujemy $\mathbb{P}$ [jest kolizja dla $h_i] \leqslant \frac{1}{2}$. To oznacza, że średnio po nie więcej niż dwóch próbach znajdziemy taką funkcję h_i, która nie ma kolizji. Czas sprawdzenia pojedynczej funkcji h_i to $O(|S_i|^2)$ dla inicjalizacji h_i oraz $O(|S_i|)$ dla whaszowania S_i i sprawdzenia, czy nie ma kolizji. Tak więc oczekiwany czas drugiej fazy to $O(\sum_{i \in \{1,\ldots,n\}} |S_i|^2)$.

Analiza Czas wyszukiwania to:

- obliczenie $h(x)$,
- obliczenie $h_{h(x)}(x)$,
- zajrzenie do $T_{h(x)}[h_{h(x)}(x)]$.

Czyli czas wyszukiwania jest pesymistycznie stały.

Na rozmiar struktury składa się tablica pierwszego poziomu rozmiaru $O(n)$ oraz tablice drugiego poziomu rozmiaru $O(\sum_{i \in \{1,\ldots,n\}} |S_i|^2)$. Tę wielkość można oszacować następująco:

$$\sum_{i \in \{1,\ldots,n\}} |S_i|^2 = 2 \sum_{i \in \{1,\ldots,n\}} \binom{|S_i|}{2} + n \leqslant 3n, \tag{2}$$

ponieważ $\sum_{i \in \{1,\ldots,n\}} \binom{|S_i|}{2}$ jest całkowitą liczbą kolizji elementów z S przy użyciu funkcji h, a pamiętamy, że h została wybrana w taki sposób, że liczba kolizji nie przekracza n. Rozmiar jest zatem rzędu $O(n)$.

Oczekiwany czas wykonania pierwszej fazy jest rzędu $O(n)$. Z nierówności (2) wynika, że oczekiwany czas wykonania drugiej fazy również jest $O(n)$.

3 Konstrukcje rodzin uniwersalnych i niezależnych

3.1 Rodzina 2-uniwersalna

Niech p będzie dowolną liczbą pierwszą większą od $|U|$.

Pokażemy, że rodzina

$$\mathbb{H} = \{x \mapsto [(ax + b) \mod p] \mod m \mid a \in \{1, \ldots, p-1\}, b \in \{0, \ldots, p-1\}\},$$

jest 2-uniwersalna. Weźmy $x \neq y$ należące do U. Oznaczmy

$$\begin{aligned} x' &:= (ax + b) \mod p \\ y' &:= (ay + b) \mod p \end{aligned}$$

Dzięki temu, że p jest pierwsza, $\mathbb{Z}_p$ jest ciałem, a więc

$$ax + b \equiv_p ay + b \iff x \equiv_p y,$$

czyli zawsze $x' \neq y'$, bo $x, y \in U$, a $|U| < p$.

Pokażemy teraz, że dla dowolnych $i \neq j$ ze zbioru $\{0, \ldots, p-1\}$ zachodzi

$$\mathbb{P}\left[x' = i \wedge y' = j\right] = \frac{1}{p(p-1)}, \tag{3}$$

a więc para (x', y') jest losową parą uporządkowaną różnych liczb z $\mathbb{Z}_p$. Zbiór funkcji haszujących $\mathbb{H}$ rozmiaru $p(p-1)$ jest naszą przestrzenią probabilistyczną. Ile jest zdarzeń elementarnych (funkcji haszujących $h \in \mathbb{H}$, czyli par (a, b)), takich że $x' = i$ i $y' = j$? Każda taka para (a, b) jest wyznaczona przez układ równań

$$\begin{cases} ax + b = i \\ ay + b = j. \end{cases}$$

Ten układ ma jednoznaczne rozwiązanie, ponieważ

$$\det \begin{bmatrix} x & 1 \\ y & 1 \end{bmatrix} \neq 0.$$

Zatem istnieje dokładnie jedna para spośród $p(p-1)$, która spełnia układ równań, zatem (3) jest udowodnione.

Teraz pokażemy, że

$$\mathbb{P}\left[x' \equiv y' \mod m\right] \leqslant \frac{1}{m}. \tag{4}$$

Jeśli to zdarzenie zachodzi, to

$$\begin{cases} x' = km + r \\ y' = lm + r. \end{cases}$$

Dla ustalonego x' istnieje co najwyżej $\lceil \frac{p}{m} \rceil - 1$ liczb $l \neq k$, które dadzą nam taki y', zatem sumując po wszystkich p możliwych wartościach x' dostajemy

$$\mathbb{P}\left[x' \equiv y' \mod m\right] \leqslant p\frac{\lceil \frac{p}{m} \rceil - 1}{p(p-1)} \leqslant \frac{\frac{p+m-1}{m} - 1}{p-1} = \frac{p-1}{m(p-1)} = \frac{1}{m} \tag{5}$$

Z (4) wynika, że rodzina $\mathbb{H}$ jest rzeczywiście 2-uniwersalna. Jednak nie jest ona 2-niezależna. Nie ma niezależności zmiennych $h(x)$ i $h(y)$ co wynika z (3).

Poza tym zmienna $h(x)$ nie jest jednostajna, bowiem dla $i < (p \mod m)$ mamy

$$\mathbb{P}\left[h(x) = i\right] = \sum_{k:i+km<p} \mathbb{P}\left[x' = i + km\right] = \frac{1}{p}\lceil\frac{p}{m}\rceil,$$

a ta liczba jest z przedziału $(\frac{1}{m}, \frac{2}{m})$.

3.2 Rodzina k-niezależna (prawie)

Rodzina opisana w poprzedniej sekcji nie jest 2-niezależna, ale można powiedzieć, że jest prawie 2-niezależna, ponieważ do spełnienia wymaganych równości brakowało jej stosunkowo niewiele. W tej sekcji skonstruujemy rodzinę, która będzie prawie k-niezależna w podobnym sensie.

Niech

$$\mathbb{H}^m = \{x \mapsto \Big[(a_0 + a_1 x + \ldots + a_{k-1}x^{k-1}) \mod p\Big] \mod m \mid a_i \in \{1, \ldots, p-1\}\}.$$

Zauważmy, że tym razem każdy ze współczynników $a_1, \ldots, a_{k-1}$ może być równy 0, a więc $\mathbb{H}^m$ zawiera funkcje stałe, które w kontekście zastosowań słownikowych zachowują się fatalnie! To założenie pozwoli jednak na udowodnienie eleganckiej własności k-niezależności zmiennych $h(x_i)$. Dostaniemy więc rodzinę funkcji haszujących, która pozwoli na łatwo otrzymywać dobre oszacowania na wartość oczekiwaną (np. czasu działania operacji słownikowych), ale z pewnym prawdopodobieństwem $\frac{1}{p^{k-1}}$ wylosujemy z niej bardzo złe funkcje. To prawdopodobieństwo jest bardzo małe, bo $p > |U|$, a w zastosowaniach $|U| \approx 2^{\text{długość słowa}}$, czyli 2^{16}, 2^{32} itd.

Rozważmy parami różne zmienne $x_1, \ldots, x_k$. Podobnie jak poprzednio niech $x'_i := \sum_j a_j x_i^j \mod p$ (a więc $h(x_i) = x'_i \mod m$).

Pokażemy najpierw, że dla dowolnych $y'_1, \ldots, y'_k \in \{0, \ldots, p-1\}$ zachodzi

$$\mathbb{P}\left[\bigcap_i^k x'_i = y'_i\right] = \frac{1}{p^k}. \tag{6}$$

To zdarzenie odpowiada układowi równań

$$\begin{cases} a_0 + a_1 x_1 + \ldots + a_{k-1}x_1^{k-1} & \equiv_p & y'_1 \\ a_0 + a_1 x_2 + \ldots + a_{k-1}x_2^{k-1} & \equiv_p & y'_2 \\ \qquad\qquad \vdots & & \vdots \\ a_0 + a_1 x_k + \ldots + a_{k-1}x_k^{k-1} & \equiv_p & y'_k \end{cases}$$

Macierz

$$\begin{bmatrix} 1 & x_1 & \ldots & x_1^{k-1} \\ 1 & x_2 & \ldots & x_2^{k-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_k & \ldots & x_k^{k-1} \end{bmatrix}$$

tego układu jest macierzą Vandermonde'a, a ta ma niezerowy wyznacznik dla parami różnych $x_1, \ldots, x_k$. Zatem istnieje dokładnie jedno rozwiązanie $(a_0, \ldots, a_{k-1})$ tego układu spośród p^k, a więc równość (6) jest udowodniona. Zauważmy, że w poprzedniej sekcji w analogicznej równości dostaliśmy wynik $\frac{1}{p(p-1)}$, ponieważ w definicji rodziny $\mathbb{H}$ wykluczyliśmy funkcje stałe (a było różne od 0).

W tej chwili zauważmy, że właśnie pokazaliśmy, że rodzina

$$\mathbb{H}^p = \{x \mapsto (a_0 + a_1 x + \ldots + a_{k-1}x^{k-1}) \mod p \mid a_i \in \{1, \ldots, p-1\}\}.$$

jest rodziną k-niezależną. W szczególności jeśli h wylosowano z $\mathbb{H}^p$ to zmienne losowe $h(x)$ dla $x \in U$ są k-niezależne. Z tego łatwo wynika (zachęcamy czytelnika do sprawdzenia), że również jeśli h wylosowano z $\mathbb{H}^m$ to zmienne losowe $h(x)$ dla $x \in U$ są k-niezależne. Aby *rodzina* $\mathbb{H}^m$ była k-niezależna, potrzeba jednak jeszcze, żeby funkcja h była jednostajna, a to nie do końca jest prawdą.

Analogicznie jak w poprzedniej sekcji dla ustalonych $y_1, \ldots, y_k \in \{0, \ldots, m\}$

$$\mathbb{P}\left[\bigcap_i^k h(x_i) = y_i\right] = \mathbb{P}\left[\bigcap_i^k x_i' \equiv y_i (\mod m)\right] \leqslant \left\lceil \frac{p}{m} \right\rceil^k \frac{1}{p^k}, \tag{7}$$

ponieważ dla każdego y_i istnieje co najwyżej $\lceil \frac{p}{m} \rceil$ wartości x_i', że $x_i' \equiv y_i \mod m$, a każda konkretna krotka jest losowana z prawdopodobieństwem $\frac{1}{p^k}$ na mocy równości (6). Widzimy, że nie dostaliśmy tu oszacowania przez $\frac{1}{m^k}$, którego wymaga defnicja, jednakże zwykle $m \ll p$ a więc $\lceil \frac{p}{m} \rceil^k \frac{1}{p^k}$ jest bardzo bliskie $\frac{1}{m^k}$. Jeśli np. dobierzemy p tak, aby $\frac{m-1}{p} \leqslant \frac{1}{k}$, to

$$\left\lceil \frac{p}{m} \right\rceil^k \frac{1}{p^k} \leqslant \left(\frac{p+m-1}{pm}\right)^k = \left(\frac{1 + \frac{m-1}{p}}{m}\right)^k < \frac{e}{m^k}.$$

3.3 Praktyczna rodzina $(2,2)$-uniwersalna (Dietzfelbinger)

W praktyce rozmiar uniwersum i m są potęgami dwójki: $U = \{0, \ldots, 2^k - 1\}$, $T[0 \ldots 2^l - 1]$. Wtedy $h : \{0, \ldots, 2^k - 1\} \mapsto \{0, \ldots, 2^l - 1\}$. Rodzina funkcji haszujących jest następująca:

$$\mathbb{H}_{k,l} = \{x \mapsto (ax \mod 2^k) \text{ div } 2^{k-l} \mid a \in \{0, \ldots, 2^k - 1\} \wedge a \text{ nieparzyste}\}.$$

Operacja div 2^{k-l} bierze l pierwszych (najbardziej znaczących) bitów. Implementacja funkcji z powyższej rodziny jest bardzo łatwa, gdy liczby typu `int` są z $\{0, \ldots, 2^k - 1\}$: `(a*x) >> k-l`.

Pokażemy, że $\mathbb{H}_{k,l}$ jest $(2,2)$-uniwersalna. Niech $x, y \in \{0, \ldots, 2^k - 1\}$. Załóżmy, że $x > y$ i niech h_a będzie funkcją wybraną losowo z $\mathbb{H}_{k,l}$. Chcemy pokazać, że

$$\mathbb{P}\left[h_a(x) = h_a(y)\right] \leqslant \frac{1}{2^{l-1}}. \tag{8}$$

Policzmy, ile jest takich a, dla których $h_a(x) = h_a(y)$. Ta równość jest równoważna nierówności

$$|ax \mod 2^k - ay \mod 2^k| < 2^{k-l}.$$

Niech $z = x - y$, wtedy powyższą nierówność możemy zapisać jako

$$|az \mod 2^k| < 2^{k-l}. \tag{9}$$

Z założenia $z \not\equiv 0(\mod 2^k)$ oraz a jest nieparzyste, zatem

$$az \not\equiv 0(\mod 2^k) \tag{10}$$

Warunki (9) i (10) zachodzą, gdy

$$az \mod 2^k \in \{1, \ldots, 2^{k-l} - 1\} \cup \{2^k - 2^l + 1, \ldots, 2^k - 1\} \tag{11}$$

– pierwszy zbiór jest postaci $\underbrace{0\ldots0}_{l \text{ bitów}}\underbrace{\text{coś } \neq 0}_{k-l\text{bitów}}$, a drugi $\underbrace{1\ldots1}_{l\text{bitów}}\underbrace{\text{coś } \neq 0}_{k-l\text{bitów}}$.

Niech $z = 2^s \cdot z'$, gdzie z' jest nieparzyste. Zbiór $A = \{1, 3, 5, \ldots, 2^k - 1\}$ jest grupą z mnożeniem ($\mod 2^k$). Zbiór $z' \cdot A$ jest permutacją zbioru A:

$$z'a_1 \equiv_{2^k} z'a_2 \iff z'(a_1 - a_2) \equiv_{2^k} 0 \overset{z' \perp 2^k}{\iff} a_1 - a_2 \equiv_{2^k} 0 \iff a_1 \equiv_{2^k} a_2.$$

Tak więc ilość liczb $a \in A$ spełniających (11) jest równa ilości liczb a, dla których $a \cdot 2^s \mod 2^k$ jest postaci $\underbrace{0\ldots0}_{l \text{ bitów}}\underbrace{\text{coś } \neq 0}_{k-l\text{bitów}}$ lub $\underbrace{1\ldots1}_{l\text{bitów}}\underbrace{\text{coś } \neq 0}_{k-l\text{bitów}}$. Jeśli $s \geqslant k - l$, to końcówka będzie zerowa, więc nie ma takich liczb a. Jeśli $s < k - l$, to a zaczyna się od samych 1 lub samych 0, potem wybieramy $k - l$ bitów, z których ostatni musi być równy 1, zatem a można wybrać na $2 \cdot 2^{k-l-1} = 2^{k-l}$ sposobów, co daje ostatecznie, że

$$\mathbb{P}\left[h_a(x) = h_a(y)\right] \leqslant \frac{2^{k-l}}{2^{k-1}} = \frac{1}{2^{l-1}}.$$

4 Haszowanie kukułkowe (Pagh, Rodler 2001)

Tym razem rozwiązujemy pełny problem słownika, czyli będziemy implementować wszystkie trzy operacje – *Lookup*, *Insert*, *Delete*. Będziemy używać dwóch tablic $T_1, T_2[0 \ldots m-1]$, gdzie $m \geqslant 2n$, których elementy są ze zbioru $\{0, \ldots, |U| - 1\}$. Algorytm korzysta z dwóch funkcji haszujących $h_1, h_2 : U \mapsto \{0, \ldots, m-1\}$ wybranych z rodziny *n-niezależnej* (potem osłabimy to założenie).

W trakcie działania algorytmu będzie zachodził następujący niezmiennik:

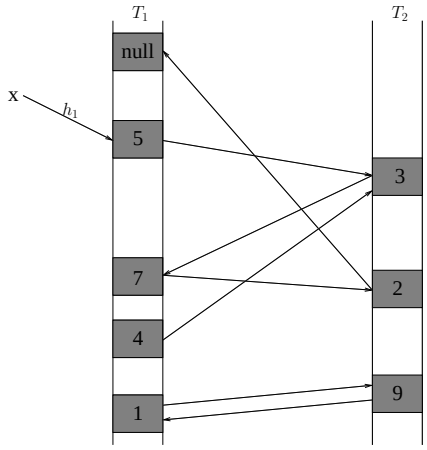
$$x \in S \iff T_1[h_1(x)] = x \vee T_2[h_2(x)] = x.$$

Stąd widać, że *Lookup* i *Delete* działają w pesymistycznym czasie stałym. Pozostaje zdefiniować $Insert$:

$Insert(x)$:

1: **if** $Lookup(x)$ **then return**
2: **for** $i = 1$ to $MaxLoop$ **do**
3: $\quad x \leftrightarrow T_1[h_1(x)]$
4: $\quad$ **if** $x =$ null **then return**
5: $\quad x \leftrightarrow T_2[h_2(x)]$
6: $\quad$ **if** $x =$ null **then return**
7: $rehash(x)$

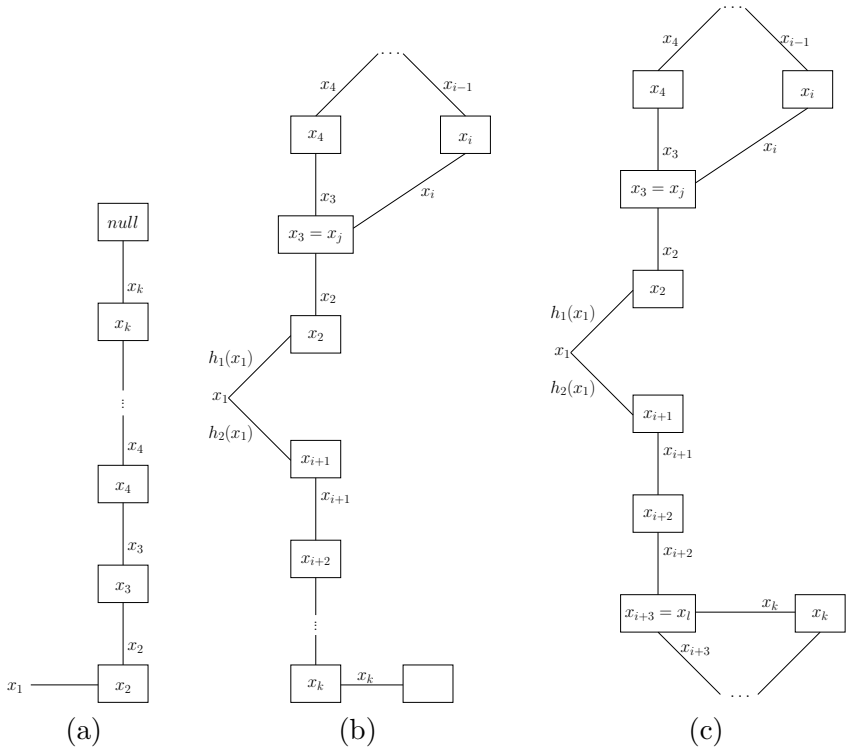
Operacja $\leftrightarrow$ zamienia wartości zmiennych, tzn. $x \leftrightarrow y$ odpowiada trzem operacjom: $a \leftarrow x$, $x \leftarrow y$, $y \leftarrow a$. Stała $MaxLoop$ jest $\leqslant n$, dokładniej wyznaczymy ją później. Po $MaxLoop$ krokach funkcja $Insert$ „poddaje się", tzn. uznaje, że z aktualnie wylosowanymi h_1 i h_2 nie jest w stanie wstawić do tablic T_1 i T_2 wszystkich elementów z $S \cup \{x\}$. W takiej sytuacji wykonywana jest operacja $rehash(x)$, która działa następująco. Tablice są czyszczone, losowane są nowe funkcje i przy ich pomocy wszystkie elementy $S \cup \{x\}$ wstawiane są na nowo do tablic za pomocą algorytmu $Insert$. Oczywiście nawet po wylosowaniu nowych h_1 i h_2 z pewnym prawdopodobieństwem jedno ze wstawień może się nie udać – wtedy ponownie losowane są nowe funkcje h_1 i h_2 i tak aż do skutku.



Rysunek 1: Przykład działania funkcji $Insert$. Elementy są przemieszczane między tablicami T_1 i T_2 po krawędziach grafu dwudzielnego (zgodnie z kierunkiem strzałek)

Oszacujmy oczekiwany czas działania funkcji $Insert$. Niech G będzie grafem dwudzielnym, którego wierzchołkami są komórki tablic T_1, T_2, oraz dla każdego $x \in S$ graf G zawiera krawędź $h_1(x)h_2(x)$. Zauważmy, że dzia-

łanie funkcji $Insert$ wyznacza pewną marszrutę w tym grafie (rys. 2). Niech $x_1, \ldots, x_k$ będą kolejnymi kluczami, które „odwiedza" ta marszruta. Marszruta ta może zawierać cykle, nie może być jednak zupełnie dowolna. Mianowicie, zobaczmy co się dzieje gdy marszruta po raz pierwszy powraca do wierzchołka, w którym już była, tzn. pewien klucz x_i jest wstawiany w miejsce klucza x_j, dla pewnego $j < i$. Wówczas x_j jest wstawiany w miejsce x_{j-1}, x_{j-1} w miejsce x_{j-2} itd, czyli cofamy się wzdłuż marszruty aż do komórki $T_1[h_1(x_1)]$. Następnie odwiedzana jest komórka $T_2[h_2(x_1)]$. Od tej chwili marszruta ponownie może odwiedzać nowe klucze. Jeśli w pewnej chwili natrafi na pustą komórkę, operacja $Insert$ się zakończy. W takiej sytuacji powiemy, że mamy do czynienia z *pojedynczym cyklem*. Jeśli natomiast ponownie natrafi na wcześniej odwiedzoną komórkę x_l, to tak generowana marszruta bez końca będzie już poruszać się po krawędziach odpowiadających odwiedzonym kluczom (a dokładniej poruszałaby się bez końca, gdyby nie sztywne ograniczenie $2MaxLoop$ na liczbę wykonanych kroków). Taką marszrutę nazwiemy *podwójnym cyklem*.



Rysunek 2: Trzy możliwe sytuacje podczas wykonywania $Insert$: ścieżka (a), pojedyńczy cykl (b), podwójny cykl (c).

Przypadek I Nie było podwójnego cyklu:

- w ogóle nie było cyklu; każdy wierzchołek był odwiedzany raz
- był tylko pojedynczy cykl

Jeśli dana marszruta ma długość k, to istnieje podmarszruta o początku w $T_1[h_1(x_1)]$ lub $T_2[h_2(x_1)]$ o długości $k/3$, która jest ścieżką. Oznaczmy przez $x'_1, \ldots, x'_{k/2}$ jej kolejne klucze (jest to spójny podciąg ciągu $x_1, \ldots, x_k$). Oszacujemy prawdopodobieństwo takiego zdarzenia:

$$\mathbb{P}\left[\text{marszruta ma długość } \geqslant k\right] \tag{12}$$

$$\leqslant 2\mathbb{P}\left[\begin{array}{l}\exists \dfrac{k}{3} \text{ parami różnych } x_2, \ldots, x_{\frac{k}{3}} \text{ t.ż.} \\ h_1(x_1) = h_1(x_2) \wedge h_2(x_2) = h_2(x_3) \wedge h_1(x_3) = h_1(x_4) \wedge \ldots\end{array}\right] \tag{13}$$

$$\leqslant 2 \cdot n^{\frac{k}{3}-1} \cdot \left(\frac{1}{m}\right)^{\frac{k}{3}-1} = 2 \cdot \left(\frac{n}{m}\right)^{\frac{k}{3}-1} \leqslant \left(\frac{1}{2}\right)^{\frac{k}{3}-2}. \tag{14}$$

Możemy teraz oszacować średnią długość marszruty:

$$\mathbb{E}\left[\text{długość ścieżki}\right] = \sum_{k \geqslant 1} \mathbb{P}\left[\text{ścieżka ma długość } \geqslant k\right] \leqslant \sum_{k \geqslant 1} \left(\frac{1}{2}\right)^{\frac{k}{3}-2} = O(1).$$

Przypadek II W funkcji *Insert* dostaliśmy podwójny cykl. Niech x_i, x_j i x_l będą takie jak w definicji podwójnego cyklu.

Dla ustalonego klucza początkowego x_1, oszacujmy liczbę możliwych cykli podwójnych odwiedzających k kluczy:

- na co najwyżej n^{k-1} sposobów wybieramy pozostałe klucze
- na k^3 sposobów wybieramy kształt cyklu (czyli indeksy i, j, l)
- na m^{k-1} sposobów wybieramy wierzchołki grafu, na których będzie leżał cykl ($k-1$, bo dla jednego klucza nie ma miejsca)

Razem co najwyżej $n^{k-1} \cdot k^3 \cdot m^{k-1}$ cykli. Każdy taki cykl składa się z k krawędzi, z których każda pojawia się niezależnie z prawdopodobienstwem $\left(\frac{1}{m}\right)^2$, na mocy niezależności losowania h_1 i h_2 oraz faktu, że h_1 i h_2 mają rozkład jednostajny. Z n-niezależności rodziny funkcji haszujących, z których wybierane są h_1 i h_2, mamy, że prawdopodobieństwo pojawienia się takiego cyklu nie przekracza $\left(\frac{1}{m}\right)^{2k}$ (zauważmy, że potrzebowaliśmy jedynie k-niezależności, a $k \leqslant 2MaxLoop$). Stąd

$$\mathbb{P}\left[\exists \text{cykl podwójny zawierający } k \text{ różnych kluczy}\right] \leqslant n^{k-1} \cdot k^3 \cdot m^{k-1} \cdot \frac{1}{m^{2k}}$$

Tak więc

$$\begin{aligned}&\mathbb{P}\left[\exists\text{cykl podwójny}\right]\\ &\leqslant \sum_{k\geqslant 3}\frac{n^{k-1}\cdot k^3\cdot m^{k-1}}{m^{2k}} \leqslant \sum_{k\geqslant 3} k^3\cdot\frac{n^{k-1}}{m^{k+1}} \leqslant \frac{1}{m^2}\sum_{k\geqslant 3} k^3\cdot\frac{1}{2^{k-1}} = O(\frac{1}{m^2}) = O(\frac{1}{n^2})\end{aligned}$$

Funkcja $Insert$ wywołuje $rehash$, gdy:

- był podwójny cykl – z prawdopodobieństwem $O(\frac{1}{n^2})$
- była długa marszruta – z prawdopodobieństwem $O(\left(\frac{1}{2}\right)^{\frac{MaxLoop}{3}})$,

jeśli więc weźmiemy $MaxLoop = 6\lg n$, to funkcja $rehash$ wywoła się z prawdopodobieństwem $O(\frac{1}{n^2})$. Oczekiwany czas funkcji $Insert$ pod warunkiem, że nie było $rehash$ jest stały, skoro średnia długość ścieżki jest stała. Jaki jest oczekiwany czas $rehash$?

$$\mathbb{P}\left[\text{któryś } Insert \text{ się nie udał}\right] = n\mathbb{P}\left[\text{konkretny } Insert \text{ się nie udał}\right] = O(\frac{1}{n}),$$

zatem oczekiwana liczba powtórzeń $rehash$ zanim wszystkie operacje $Insert$ się udadzą jest stała, stąd

$$\mathbb{E}\left[\text{czas } rehash \mid \text{był } rehash\right] = O(n)\cdot O(1) = O(n).$$

Ostatecznie

$$\begin{aligned}&\mathbb{E}\left[\text{czas } Insert\right]\\ &= \mathbb{E}\left[\text{czas } Insert \mid \text{nie było } rehash\right]\mathbb{P}\left[\text{nie było } rehash\right]\\ &+\mathbb{E}\left[\text{czas } Insert + \text{czas } rehash \mid \text{był } rehash\right]\mathbb{P}\left[\text{był } rehash\right]\\ &= O(1) + O(n)O(\frac{1}{n^2}) = O(1).\end{aligned}$$

Zauważmy na koniec, że nie potrzebujemy rodziny n-niezależnej, ale $2MaxLoop$-niezależnej, bo rozpatrujemy tylko ścieżki maksymalnie tej długości, $MaxLoop = O(\log n)$. Istnieją rodziny funkcji haszujących które oferują taką niezależność, zachowując równocześnie stały czas obliczania wartości funkcji i rozmiar $O(n)$, jednakże mają one znaczenie tylko teoretyczne. Z drugiej strony, haszowanie kukułkowe w praktyce zachowuje się dobrze dla rodzin funkcji haszujących o dużo słabszych własnościach.

Podziękowanie

Serdecznie dziękuję Markowi Adamczykowi za sporządzenie tych notatek. Łukasz Kowalik.