Wykład 4

Tablice z haszowaniem

Wprowadzenie

- **Tablice z adresowaniem bezpośrednim**
- **Tablice z haszowaniem:**
  - Adresowanie otwarte
  - Haszowanie łańcuchowe
- **Funkcje haszujące (mieszające)**
- **Haszowanie uniwersalne**
- **Idealne haszowanie**

Literatura

- Cormen, Leiserson, Rivest, “Wprowadzenie do algorytmów”, rozdz. 12

Po co to wszystko?

- **Wiele zadań wymaga operacji na tablicach: obsługi tablicy symboli (słownika) z dostępem w czasie *O*(1) w losowym przypadku.**
  Klucze nie muszą być uporządkowane.
- **Przykłady:**
  - Słowa kluczowe dla kompilatora (statyczne)
  - Numery identyfikacyjne klientów w bazie zamówień
- **Chcemy, aby struktura danych (ADT)obsługiwała operacje Search, Insert i Delete w czasie średnim *O*(1) bez zakładania niczego o elementach.**
- **Drzewa poszukiwań (BST etc.) wymagają relacji porządku i dają czas *O*(lg *n*).**
- **Tablice z haszowaniem pozwalają na dowolne klucze i dają średni czas *O*(1).**

Tablice z adresowaniem bezpośrednim

- **Korzystamy z tablicy o rozmiarze zgodnym z możliwym zakresem kluczy**
- **Wykorzystujemy klucz *k* jako indeks w tablicy *A*, (*k* → *A*[*k*])**
- **Rozmiar tablicy z haszowaniem jest proporcjonalny do zakresu kluczy, nie do ilości elementów.**

- **Potrzeba bardzo dużo pamięci!**

Tablice z haszowaniem - przegląd

- **Uogólnienie tablic $A[0..n-1]$.**
- **Zamiast używać klucza k jako indeksu w tablicy A, obliczamy indeks jako wartość pewnej funkcji haszowania (mieszania) $h(k)$:**

  $$A[k] \rightarrow A[h(k)]$$

- **Rozmiar tablicy z haszowaniem jest proporcjonalny do ilości elementów, nie do ich zakresu.**
- **Funkcja mieszająca $h(k)$ nie musi być 1-na-1.**
- **Potrzeba wtedy mechanizmu do rozwiązywania kolizji. Dwa klucze k_1 i k_2 są w kolizji, jeśli $h(k_1) = h(k_2)$.**

Przykłady

- **Słowa kluczowe dla języka programowania**
  - [**for, if, then,** …] → *h*(**for**) = 0; *h*(**if**) = 1; …
  - Rozmiar tablicy jest stały
- **Tablice rejestracyjne**
  - Numery są ustalone (format), nie wszystkie muszą być wykorzystywane
  - Można poszukiwać przy wykorzystaniu tablicy z haszowaniem o ustalonym rozmiarze.
  - Funkcja haszująca: numer rejestracyjny modulo maksymalny rozmiar tablicy z haszowaniem→ h(EL55080) = 55080 mod 10000
- **„Loginy" i hasła użytkowników systemu.**

Formalnie

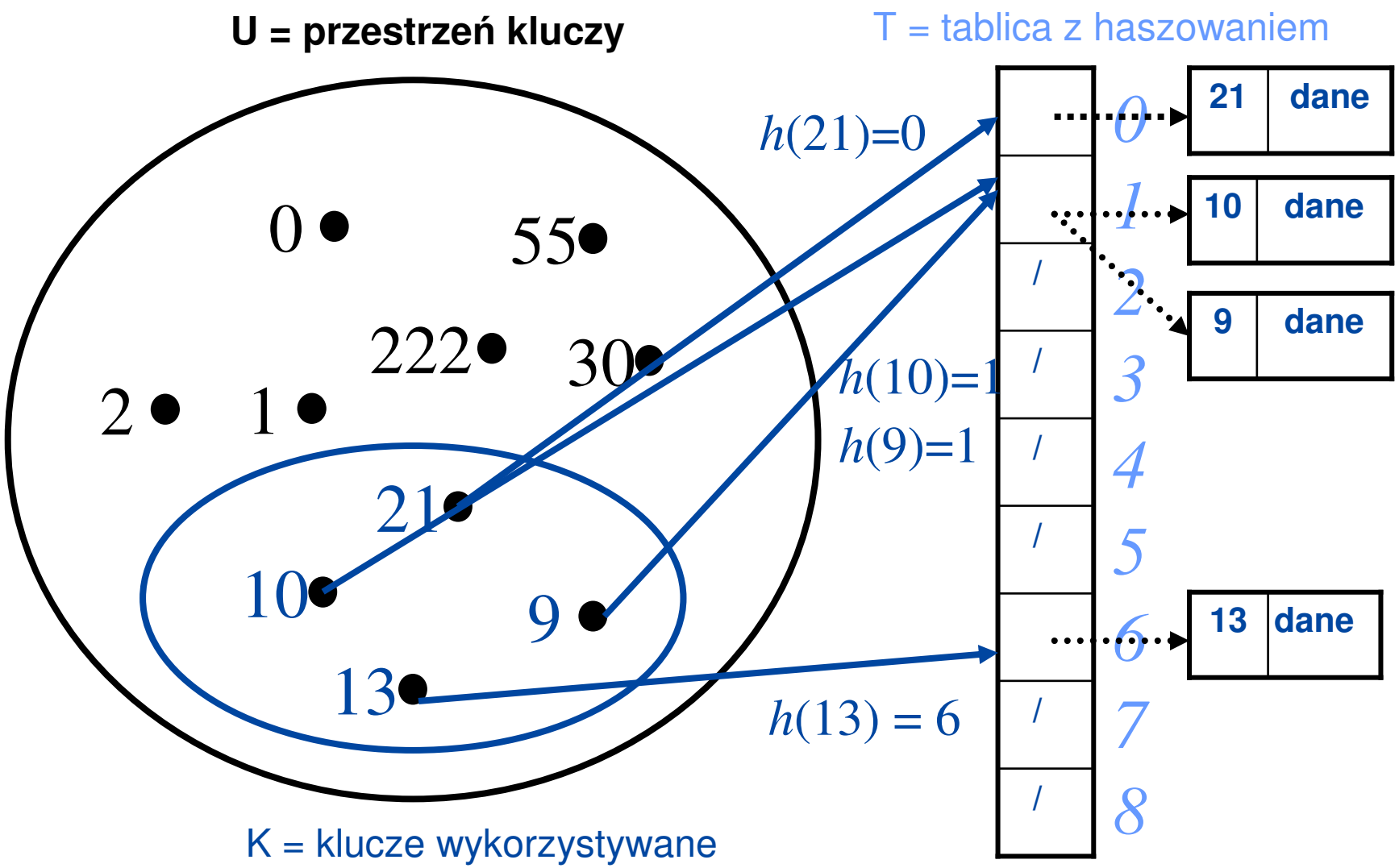
- Niech U będzie zbiorem wszystkich możliwych kluczy o rozmiarze $|U|$, K – aktualny zbiór kluczy o rozmiarze n, T tablica z haszowaniem o rozmiarze $O(m)$, $m \leq |U|$.
- Nech $h(k)$ będzie funkcja haszującą:

  $$h(k): U \rightarrow [0..m-1]$$

  mapująca klucze z U na indeksy tablicy T.

  wartość $h(k)$ można obliczyć w czasie $O(|k|) = O(1)$.
- Elementy tablicy $T[i]$ są dostępne w czasie $O(1)$. $T[i]$ = *null* jest wejściem pustym.
- Dla uproszczenia można przyjąć $U = \{0, \ldots, N-1\}$.

Haszowanie


U = przestrzeń kluczy
T = tablica z haszowaniem
0
55
222
30
2
1
21
10
9
13
K = klucze wykorzystywane
h(21)=0
h(10)=1
h(9)=1
h(13) = 6
0
1
2
3
4
5
6
7
8
21
dane
10
dane
9
dane
13
dane

Pytania

- **Jakie funkcje mieszające są dobre?**
- **Jak postępować w przypadki kolizji?**
- **Jakie założenia są niezbędne dla osiągnięcia czasu *O*(1) w średnim przypadku?**

Co z przypadkiem worst-case?

Główne podejścia:

- Adresowanie bezpośrednie
- Adresowanie otwarte
- Łańcuchy

Wyzwania: dobra funkcja mieszająca, uniwersalne haszowanie.

Adresowanie bezpośrednie, jako haszowanie

- **Tablica z haszowaniem ma rozmiar *m*, *T* [0..*m*–1].**
- **Ilość wykorzystywanych kluczy *n* jest bliska *m*, lub *m* jest mała (w porównaniu z dostępnymi zasobami - pamięcią).**
- **Klucz *k* staje się indeksem *T*, tj. funkcja mieszająca jest następująca: *h*(*k*) = *k*.**
- **Nie ma kolizji, czas dostępu wynosi *O*(1), w najgorszym wypadku.**
- **Nie ma potrzeby przechowywania klucza – tylko dane.**
- **Problemy:**
  - Metoda ta wymaga dużo pamięci
  - Niemożliwa do zrealizowania dla dużych *m*

Tablica z haszowaniem

- **Możemy używać funkcji mieszającej wiele-do-jednego *h*(*k*) do mapowania kluczy *k* na indeksy *T*.**
- **Zbiór wykorzystywanych kluczy *K* może być dużo mniejszy od przestrzeni kluczy *U*, tj. *m* « |*U*|.**
- **Rozwiązywanie kolizji kiedy $h(k_1) = h(k_2)$ dwoma metodami:**
  I. Adresowanie otwarte
  II. Łańcuchy

Współczynnik zapełnienia

Założenie o *równomiernym haszowaniu*: dla każdego klucza wszystkich *m!* permutacji zbioru *{0,1,…,m-1}* jest jednakowo prawdopodobnych jako ciąg kolejnych próbkowań (ciąg kontrolny).

Definicja: *współczynnikiem zapełnienia* α dla tablicy z haszowaniem *T* z *m* pozycjami określamy stosunek *n*/*m*, gdzie *n* jest ilością przechowywanych elementów.

Jest to średnia liczba elementów przechowywanych w jednej pozycji tablicy

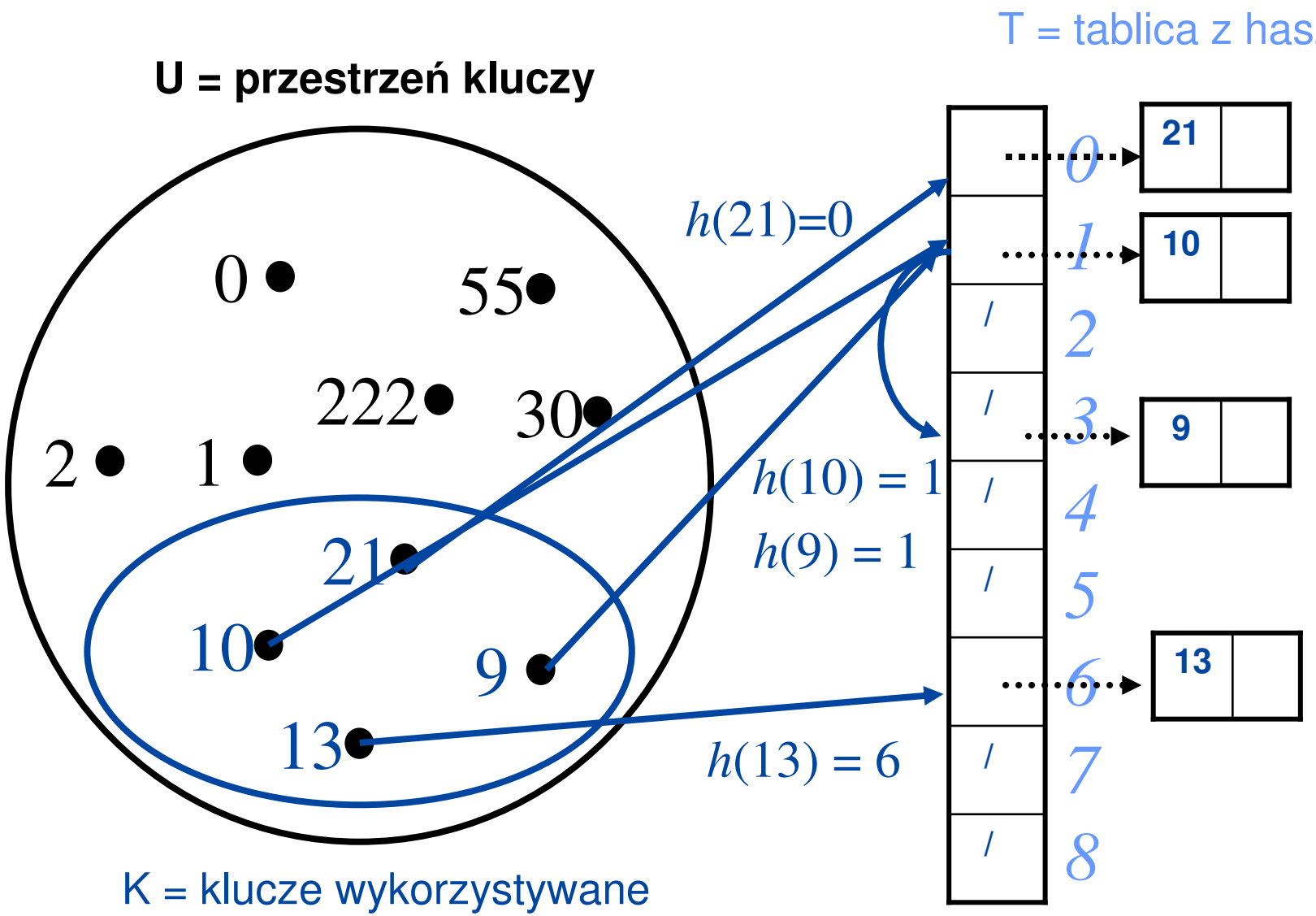
I. Rozwiązywanie kolizji metodą adresowania otwartego

- **Wszystkie klucze k_i mapowane w ta samą pozycję $T[h(k_i)]$ są wstawiane bezpośrednio do tablicy w pierwsze wolne miejsce.**
- **Poszczególne pozycje tablicy mogą zawierać elementy, albo wartości *null*.**
- **Funkcja haszująca powinna być zmodyfikowana (dwuargumentowa, gdzie drugi argument jest numerem próby) :**

  $h(k,i)$: $U \times [0..m-1] \rightarrow [0..m-1]$

  Sekwencja kolejnych miejsc w tablicy dla elementu: $h(k,0)$, $h(k,1)$…

Adresowanie otwarte



Operacje potrzebne dla adresowania otwartego

- **Insert: testujemy kolejne miejsca w tablicy, aż do odnalezienia wolnego. Sekwencja kolejnych miejsc do sprawdzenia zależy od wstawianego klucza. Jeśli nie odnajdujemy wolnego miejsca po *m* próbach, oznacza to że tablica jest pełna.**
- **Search: testujemy tę samą sekwencję pozycji co przy wstawianiu, albo do momentu odnalezienia poszukiwanego klucza (sukces), albo natrafienia na wolną pozycję (porażka).**
- **Delete: klucze nie mogą być zwyczajnie usuwane! Zamiast tego można oznaczać pozycje, jako „usunięte”. Procedura poszukiwania powinna kontynuować działanie przy natrafieniu na taką wartość, natomiast procedura wstawiania traktować ją tak jak *nill*.**

Złożoność: zależna od długości sekwencji próbkowania.

Strategie próbkowania

Najczęściej stosuję się jedną z trzech strategii:

1. Adresowanie liniowe
2. Adresowanie kwadratowe
3. Haszowanie dwukrotne

Adresowanie liniowe

Funkcja haszująca:

$$h(k, i) = (h'(k) + i) \bmod m$$

gdzie $h'(k)$ jest zwykłą funkcją haszującą (niezależną od numeru próby).

Dla klucza k, sekwencja próbkowania jest wtedy następująca:

$T[h'(k)], T[h'(k)+1], \ldots T[m-1], T[0], T[1] \ldots T[h'(k)-1]$

Problem: tendencja do *grupowania zajętych pozycji (primary clustering)*.

Długie, spójne ciągi zajętych pozycji szybko się powiększają, co spowalnia operacje poszukiwania.

Adresowanie kwadratowe

Funkcja haszująca:

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

gdzie c_1 i c_2 stałe $\neq 0$, $h'(k)$ zwykła funkcja haszującą (niezależna od numeru próby).

W przeciwieństwie do adresowania liniowego, kolejne rozpatrywane pozycje są oddalone od początkowej o wielkość zależną od kwadratu numeru próby *i*.

Prowadzi to do mniej groźnego zjawiska grupowania – określanego, jako *grupowanie wtórne* (*secondary clustering)*: klucze o tej samej pozycji początkowej dają takie same sekwencje.

Haszowanie podwójne

Funkcja haszująca:

$$h(k, i) = (h_1(k) + ih_2(k)) \bmod m$$

gdzie $h_1(k)$ i $h_2(k)$ dwie funkcje haszujące.

Pierwsza sprawdzana pozycja to $T[h_1(k)]$. Kolejne próby są oddalone od początkowej o $h_2(k)$ mod m.

Wartość $h_2(k)$ powinna być względnie pierwsza z rozmiarem tablicy m, aby mieć gwarancję że cała tablica zostanie przeszukana.

Przykłady funkcji:

$$h_1(k) = k \bmod m$$

$$h_2(k) = 1 + (k \bmod m') \qquad \text{gdzie } m' < m \text{ (np. } m-1, m-2)$$

Generuje $\Theta(m^2)$ istotnie różnych sekwencji (dla liniowego i kwadratowego $\Theta(m)$)

Analiza adresowania otwartego

Twierdzenie: jeśli współczynnik zapełnienia tablicy z haszowaniem wynosi $\alpha = n/m < 1$, to oczekiwania liczba porównań kluczy w czasie wyszukiwania elementu (przy spełnieniu założeniu o równomiernym haszowaniu) :

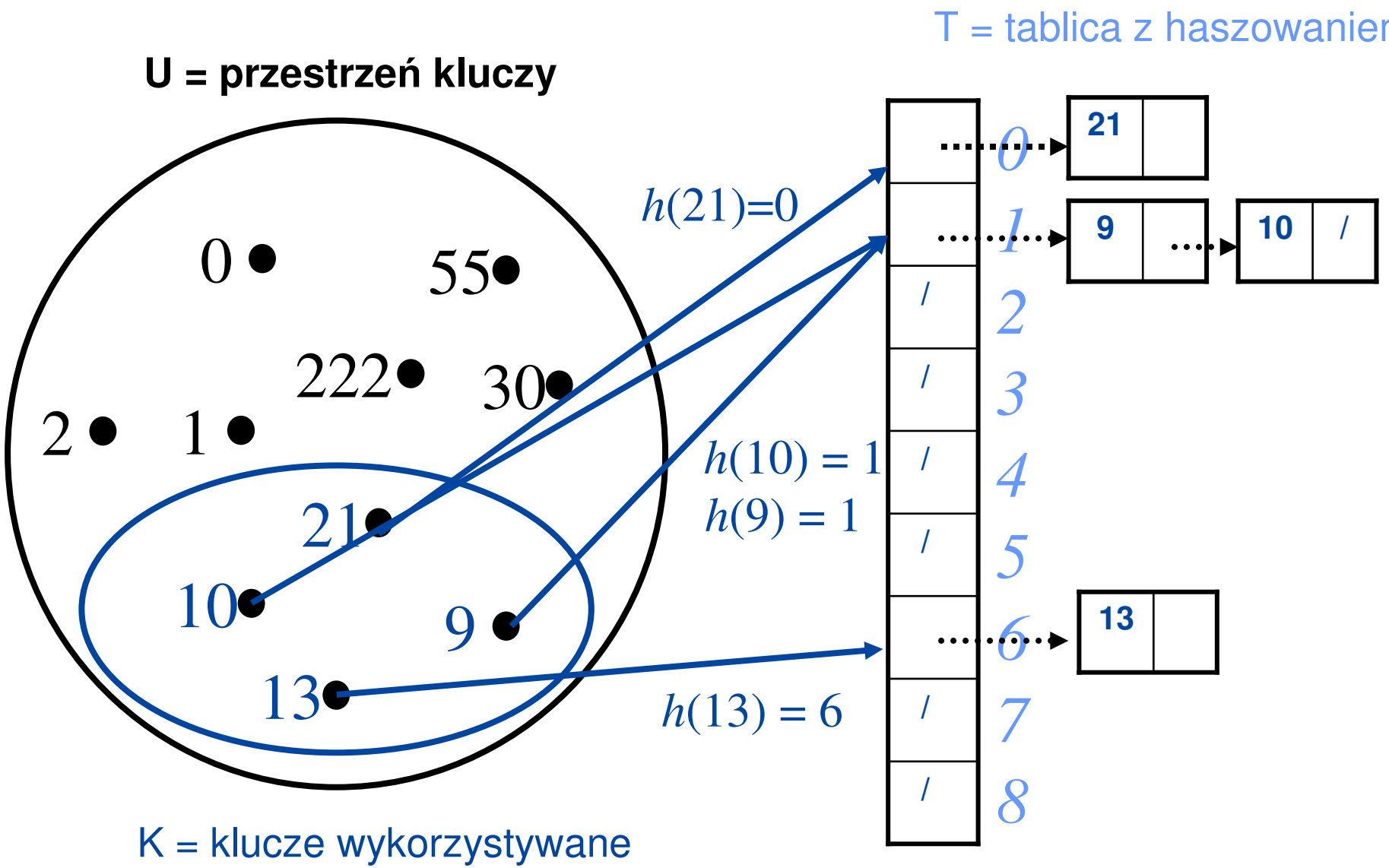
- co najwyżej $1/(1-\alpha)$ dla nieudanego poszukiwania
- co najwyżej $1/\alpha \ln 1/(1-\alpha)$ dla udanego poszukiwania

- **Jeśli α jest stałe, to czas poszukiwania wynosi $O(1)$**

II. Rozwiązywanie kolizji metodą łańcuchową

- **Wszystkie klucze k_i , które trafiają w tę samą pozycję $T[h(k_i)]$ umieszcza się w liście liniowej L_j , $j = h(k_i)$.**
- **Elementy tablicy przechowują wskaźniki do list L_j**
- **Wstawianie (Insert): nowy klucz wstawiany jest na początek listy L_j (czas $O(1)$).**
- **Poszukiwanie/usuwanie (Search/Delete): przeszukujemy listę w poszukiwaniu klucza (czas proporcjonalny do ilości elementów najdłuższej listy).**

Metoda łańcuchowa



Proste równomierne haszowanie

Proste równomierne haszownie oznacza, że losowo wybrany element z jednakowym prawdopodobieństwem trafia na każdą z *m* pozycji, niezależnie od tego gdzie trafiają inne elementy.

Twierdzenie: w tablicy haszowania wykorzystującej łańcuchową metodę rozwiązywania kolizji, przy założeniu prostego, równomiernego haszowania, średni czas działania procedury wyszukiwania (zarówno dla sukcesu, jak i porażki) wynosi $\Theta(1+\alpha)$, gdzie α jest współczynnikiem zapełnienia tablicy.

Dobre funkcje haszujące

- **Wydajność haszowania zależy w olbrzymiej mierze od zbioru wykorzystywanych kluczy oraz funkcji wykorzystywanej do haszowania.**
- **Dobre funkcje haszujące to takie, które spełniają założenie prostego równomiernego haszowania!**
- **Zwykle jednak nie można (lub nie jest łatwo) stwierdzić, czy założenie takie jest spełnione. Najczęściej nie znamy rozkładu prawdopodobieństwa występowania kluczy.**
- **Szukając dobrych funkcji najczęściej poszukuje się funkcji działających dobrze „w większości wypadków” → heurystyczne**

Heurystyczne funkcje haszujące

- **Funkcja powinna dobrze się sprawować dla większości zbiorów kluczy (takich, które naprawdę występują w zadaniu), niekoniecznie dla specjalnie przygotowanych „złośliwych” zbiorów kluczy.**
- **Zbiór wykorzystywanych kluczy *K* zwykle nie jest losowy, ale posiada pewne cechy (np. wspólne początkowe bity, wielokrotności pewnej liczby, itp.).**
- **Celem jest znalezienie takiej funkcji haszujacej, która tak dzieli cały zbiór potencjalnych kluczy *U* , że dla zbioru kluczy aktualnych *K* wydaje się on losowy. Przypadek najgorszy (worst-case) powinien być mało prawdopodobny.**

Haszowanie modularne

- Niech $U = N = \{0,1,2, \ldots\}$, będzie zbiorem liczb naturalnych.
- Mapujemy klucz k na pozycję m przez branie reszty z dzielenia k przez m:

$$h(k) = k \bmod m$$

- aby taka metoda działała dobrze należy unikać takich m , które są potęgami 2 ($m = 2^p$) – ponieważ wtedy wybieramy ostatnie p bitów klucza, ignorując istotna część informacji.
- <u>Heurystyka:</u> wybieramy jako m liczbę pierwszą odległą od potęg 2.

Przykład haszowania modularnego

- Weźmy $|U| = n = 2000$ i załóżmy, że dopuszczamy maksymalnie 3 kolizje dla klucza.
- Jaki powinien być rozmiar tablicy (m)?
- Mamy $floor(2000/3) = 666$; liczba pierwsza bliska tej wartości, a jednocześnie daleka od potęg 2 to np. 701.
- Stąd funkcja haszująca może być taka:

$$h(k) = k \bmod 701$$

- Wtedy klucze 0, 701, i 1402 są mapowane na 0.

Haszowanie przez mnożenie

- **Mapujemy klucz *k* na jedną z *m* pozycji przez pomnożenie go przez stałą *a* z zakresu 0 < *a* < 1, dalej wyznaczamy część ułamkową *ka*, i mnożymy ją przez *m*:**

$$h(k) = \lfloor m(ka - \lfloor ka \rfloor) \rfloor , \quad 0 < a < 1$$

- **Metoda taka jest mniej wrażliwa na wybór *m* ponieważ „losowe" zachowanie wynika z braku zależności pomiędzy kluczami a stałą *a.***
- **<u>Heurystyka:</u> wybieramy *m* jako potęgę 2 i *a* jako liczbę bliską „złotemu podziałowi" :**

$$a = \left(\sqrt{5} - 1\right)/2 = 0.6180339887...$$

Haszowanie uniwersalne

- **Idea: dobieramy funkcję haszującą *losowo* w sposób niezależny od kluczy.**
- **Funkcja wybierana jest z rodziny funkcji o pewnych szczególnych własnościach (takich, które „średnio” zachowują się dobrze).**
- **Gwarantuje to, że nie ma takiego zestawu kluczy, który zawsze prowadzi do najgorszego przypadku.**
- **Pytanie: jak określić taki zbiór funkcji?**
- **Wybieramy ze *skończonego* zbioru *uniwersalnych funkcji haszujących*.**

Haszowanie uniwersalne

- **<u>Cel</u>: chcemy aby zachodziło założenie o prostym równomiernym haszowaniu – tak aby klucze były średnio rozproszone równomiernie.**
- **<u>Właściwości prostego, równomiernego haszowania</u>:**
  - Dla dowolnych dwóch kluczy k_1 i k_2, i dowolnych dwóch pozycji y_1 i y_2, prawdopodobieństwo tego, że $h(k_1) = y_1$ i $h(k_2) = y_2$ wynosi dokładnie $1/m^2$.
  - Dla dwóch kluczy k_1 i k_2, prawdopodobieństwo kolizji, tj. $h(k_1) = h(k_2)$ wynosi dokładnie $1/m$.
- **Chcemy, aby rodzina funkcji haszujących H była tak dobrana, że szansa kolizji jest taka sama jak przy prostym, równomiernym haszowaniu.**

Haszowanie uniwersalne

Definicja: niech H będzie skończoną rodziną funkcji haszujących, mapujących zbiór dopuszczalnych kluczy U na zbiór $\{0,1,\ldots,m-1\}$.

H nazywamy *uniwersalną* jeżeli dla każdej pary różnych kluczy k_1 i k_2 z U, ilość funkcji haszujących h z H, dla których $h(k_1) = h(k_2)$ jest równa co najwyżej $|H|/m$.

- Inaczej mówiąc, jeśli losowo wybierzemy funkcję z takiej rodziny to szansa na kolizję pomiędzy różnymi kluczami k_1 i k_2 nie jest większa niż $1/m$.

Wartość oczekiwana długości listy

Twierdzenie: niech *h* będzie funkcją haszującą, wybraną losowo z uniwersalnej rodziny funkcji haszujących. Jeśli zastosujemy ją do haszowania *n* kluczy w tablicę *T* o rozmiarze *m*, to oczekiwana długość łańcucha, do którego dołączany jest klucz (przy założeniu, że $\alpha = n/m$ jest współczynnikiem zapełnienia) wynosi:

- jeśli *k* nie ma w tablicy – co najwyżej α.
- jeśli *k* jest już w tablicy – co najwyżej $1+\alpha$.

Złożoność

<u>Wniosek</u>: Przy zastosowaniu uniwersalnego haszowania i rozwiązywania kolizji metodą łańcuchową, dla tablicy o rozmiarze m, oczekiwany czas n operacji wstawiania, usuwania i wyszukiwania wynosi $\Theta(n)$.

Konstrukcja rodziny uniwersalnej

- **Wybieramy liczbę pierwszą taką, że $p > m$ i p jest większe od zakresu kluczy aktualnych K. Niech Z_p oznacza zbiór $\{0,\ldots p-1\}$, i niech a i b będą dwoma liczbami z Z_p.**
- **rozważmy funkcję:**

$$h_{a,b}(k) = (ak + b) \bmod p$$

- **Rodziną wszystkich, takich funkcji jest:**

$$H_{p,m} = \{h_{a,b} \mid a,b \in Z_p \text{ i } a \neq 0\}$$

- **Aby wybrać losową funkcję z tej rodziny – wybieramy losowo a i b ze zbioru Z_p.**

Konstrukcja rodziny uniwersalnej

- **Lemat: dla dwóch różnych kluczy k_1 i k_2, oraz dwóch liczb x_1 i x_2 z Z_p, prawdopodobieństwo, że k_1 trafi na pozycję x_1 i k_2 na pozycję x_2 wynosi $1/p^2$.**
- **Dowód: rozważmy dwa równania ze zmiennymi a i b:**

$$ak_1 + b = x_1 \bmod p$$

$$ak_2 + b = x_2 \bmod p$$

równania te zawsze mają jednoznaczne rozwiązanie jeśli p jest liczbą pierwszą! Dla dowolnych x_1 i x_2, istnieje funkcja haszująca z parametrami a i b mapująca k_1 na x_1 i k_2 na x_2.

- **Zatem, szansa wybrania takiej funkcji jest równa szansie wybrania właściwego a i b, a ta wynosi dokładnie $1/p^2$.**

Konstrukcja rodziny uniwersalnej

Ponieważ zakres kluczy może być bardzo duży, zawężamy zbiór funkcji haszujących do *m* wartości:

$$h_{a,b}(k) = ((ak + b) \bmod p) \bmod m$$

Rodzina:

$$H_{p,m} = \{h_{a,b} : = a, b \in Z_p\}$$

jest poszukiwaną rodziną uniwersalną funkcji haszujących.

Haszowanie uniwersalne - podsumowanie

- **Haszowanie uniwersalne daje czas *O*(1) w przypadku średnim i to dla dowolnego zbioru aktualnych kluczy, nawet jeśli trafiamy na same „złośliwe" układy kluczy.**
- **Szansa na złe zachowanie się metody jest bardzo mała.**
- **Jednak dla dynamicznych zbiorów kluczy, nie możemy powiedzieć z wyprzedzeniem czy dana funkcja będzie dobra, czy nie…**

Idealne haszowanie

- Haszowanie uniwersalne zapewnia średni czas $O(1)$ dla dowolnego zbioru kluczy.
- Czy można zrobić to lepiej? W niektórych przypadkach TAK!
- Idealne haszowanie zapewnia czas $O(1)$ w najgorszym przypadku (worst-case) dla statycznego zbioru kluczy (takiego, w którym klucz raz już zachowany, nie zmienia się nigdy).
- Przykłady statycznych zbiorów kluczy: słowa kluczowe dla jezyka programowania, nazwy plików na płycie CD.

Idealne haszowanie

Idea: korzystamy z dwupoziomowego schematu haszowania – za każdym razem jest to haszowanie uniwersalne.

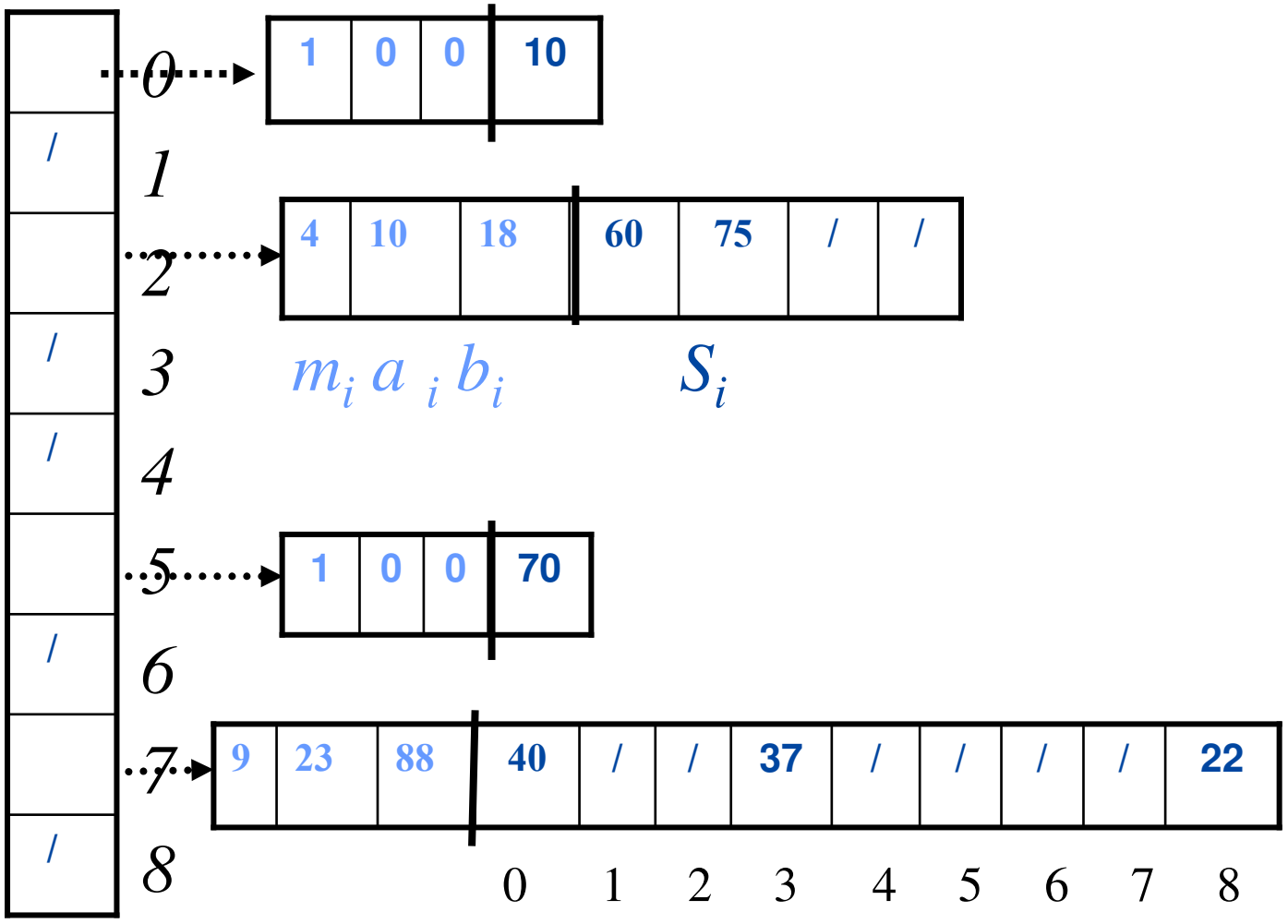
poziom 1: haszujemy łańcuchowo: n kluczy ze zbioru K jest umieszczanych na m pozycjach w tablicy T z wykorzystaniem funkcji $h(k)$ wybranej z rodziny uniwersalnej.

poziom 2: zamiast tworzyć listę liniową dla kluczy wstawianych na pozycję j, wykorzystujemy drugą tablicę z haszowaniem S_j skojarzoną z funkcją $h_j(k)$. Wybieramy $h_j(k)$ aby mieć pewność, że nie zachodzą kolizje, S_j ma rozmiar równy kwadratowi ilości n_j - kluczy umieszczonych na tej pozycji $|S_j| = n_j^2$.

Idealne haszowanie - przykład

Pierwsza funkcja mieszająca: $h(k) = ((3k + 42) \bmod 101) \bmod 9$
Druga funkcja mieszająca: $h_i(k) = ((a_i k + b_i) \bmod p) \bmod m_i$

$K = \{10,22,37,40,60,70,75\}$



Idealne haszowanie - analiza

Twierdzenie: jeśli przechowujemy n kluczy w tablicy o rozmiarze $m = n^2$ przy wykorzystaniu funkcji $h(k)$ losowo wybranej z rodziny uniwersalnej funkcji haszujących, wtedy prawdopodobieństwo kolizji wynosi $< \frac{1}{2}$.

Dowód: mamy $n(n-1)/2$ par kluczy, które mogą kolidować ze sobą, i prawdopodobieństwo kolizji dla każdej z nich wynosi $1/m$ jeśli funkcja h jest wybrana z rodziny uniwersalnej. Dla $m = n^2$ mamy:

$$\frac{n(n-1)}{2}\frac{1}{m} = \frac{n^2-n}{2n^2} \leq \frac{n^2}{2n^2} = \frac{1}{2}$$

Zatem dla każdej pary kluczy istnieje większe prawdopodobieństwo, że NIE będzie między nimi kolizji!

Idealne haszowanie - analiza

- **Dla dużych *n*, przechowywanie tablicy o rozmiarze $m = n^2$ jest kosztowne.**
- **Dla zredukowania potrzebnej pamięci można wykorzystać następujący schemat:**
  - poziom 1: *T* rozmiaru $m = n$
  - poziom 2: S_j rozmiaru $m_j = n_j^2$
- **Jeśli mamy pewność, że dla tablic drugiego poziomu nie ma kolizji, czas dostępu (również worst-case) jest stały.**
- **Pytanie – jaki jest oczekiwany rozmiar potrzebnej pamięci *?***

Idealne haszowanie – analiza pamięci

- **Rozmiar tablicy pierwszego poziomu - $O(n)$.**
- **Oczekiwany rozmiar wszystkich tablic drugiego poziomu wynosi:**

$$\sum_{j=1}^{m} n_j^2 = \sum_{j=1}^{m} (n_j + 2\frac{n_j(n_j-1)}{2}) =$$

$$= n + 2\sum_{j=1}^{m} (\frac{n_j(n_j-1)}{2})$$

- **Wyrażenie w sumie określa łączną ilość kolizji.**
- **Średnio jest to $1/m$ razy ilość par. Ponieważ $m = n$, daje to co najwyżej $n/2$.**
- **Stąd, oczekiwana rozmiar tablic drugiego poziomu wyniesie mniej niż $2n$.**

Podsumowanie

- **Haszowanie jest uogólnieniem abstrakcyjnego typu danych dla tablicy.**
- **Pozwala na stały czas dostępu i liniowe składowanie dla dynamicznych zbiorów kluczy.**
- **Kolizje rozwiązywane są poprzez *metodę łańcuchową* albo *otwarte adresowanie*.**
- **Haszowanie uniwersalne zapewnia gwarancję oczekiwanego czasu dostępu.**
- **Idealne haszowanie zapewnia gwarancję czasu dostępu w przypadku statycznych zbiorów kluczy.**
- **Haszowanie nie jest dobrym rozwiązaniem przy poszukiwaniach związanych z porządkiem (szukanie maksimum, następnika itp.) – nie ma porządku pomiędzy kluczami w tablicy.**