STRUKTURY DANYCH I ZŁOŻONOŚĆ OBLICZENIOWA

Część 9

Mieszanie, tablice mieszające

WYSZUKIWANIE ELEMENTU W ZBIORZE

Do najczęściej wykonywanych operacji na zbiorze elementów należą:

- **wstawianie elementu do zbioru,**
- **usuwanie elementu ze zbioru,**
- **wyszukiwanie elementów o zadanych własnościach (np. elementu „najmniejszego" lub elementu o wskazanej wartości „klucza").**

Struktura danych umożliwiająca wykonywanie takich operacji nazywa się *słownikiem*.

***Przykłady słowników*:**

- **bazy danych**
- **tablice identyfikatorów w kompilatorach**
- **komputerowe słowniki języków naturalnych**

***Zagadnienie słownika* :**

Podać strukturę danych umożliwiającą wykonywanie następujących operacji na skończonym zbiorze *S*:

- ***construct(S) :: S* = ∅ (inicjacja pustego słownika)**
- ***search(x, S) ::* (sprawdzenie, czy element *x* ∈ *S*; jeśli tak, to wskazanie miejsca, w którym znajduje się *x*)**
- ***insert(x, S) :: S = S* ∪ *{x}* (wstawienie elementu *x* do słownika)**
- ***delete(x, S) :: S = S - {x}* (usunięcie elementu *x* ∈ *S* ze słownika)**

PRZYKŁADY STRUKTUR WYKORZYSTYWANYCH DO IMPLEMENTACJI SŁOWNIKÓW

- Listy (liniowe)
- Listy z przeskokami
- Drzewa poszukiwań binarnych (AVL, czerwono-czarne, rozchylane,…)
- Wielokierunkowe drzewa poszukiwań (B-drzewa, drzewa „trie”, R-drzewa,…)

Złożoność operacji wyszukiwania elementu o wskazanym „kluczu” - O(lg N) … O(N)

- Kopce binarne (klasyczne, minimaksowe, sprzężone,…)
- Kopce dwumianowe
- Kolejki priorytetowe

Złożoność operacji wyszukiwania elementu „najmniejszego”/”największego” - O(1)

Problem:

Czy istnieją struktury danych umożliwiające wyszukiwanie elementu o wskazanym „kluczu” w procedurze o złożoności O(1)?

MIESZANIE (*ang.: hashing*)

Geneza:

Typem struktury danych o dostępie swobodnym jest typ tablicowy.

Gdyby typem indeksującym dla typu tablicowego był typ składowej kluczowej elementu danych, to procedura wyszukiwania elementu o określonej wartości składowej kluczowej byłaby procesem o złożoności O(1).

Z reguły moc typu składowej kluczowej „wielokrotnie" przewyższa liczbę elementów danych, które zgodnie z przewidywaniami mogą się znaleźć w słowniku podczas jego „istnienia".

Przykład: 8-znakowe „hasła" (kody uwierzytelniające) składające się z samych cyfr wykorzystywane w „populacji" liczącej 1000 podmiotów (relacja $10^8 \leftrightarrow 10^3$).

Jeżeli zastosuje się funkcję przetwarzającą w czasie O(1) dowolną wartość składowej kluczowej na jedną z wartości typu indeksującego typ tablicowy, przy czym moc $\mathscr{I}$ typu indeksującego będzie „wielokrotnie" mniejsza od mocy $\mathscr{K}$ typu składowej kluczowej, to można uzyskać pożądany efekt, czyli złożoność operacji wyszukiwania O(1) bez konieczności tworzenia tablicy $\mathscr{K}$-elementowej.

Taka funkcja nosi nazwę funkcji transformującej klucze albo funkcji mieszającej (*ang.: hash function*), zaś odpowiednie tablice, to tablice mieszające (*ang.: hash tables*).

Pożądane własności funkcji mieszającej (transformującej klucze)

Funkcja mieszająca *H* jest funkcją odwzorowującą pewien zbiór składowych kluczowych elementów zbioru *S* w zbiór adresów *0 ÷ (m-1)* dla pewnej liczby naturalnej *m* = $\mathscr{I}$. „Dobra" funkcja mieszająca powinna być:

- ***łatwo obliczalna (w szczególności - jednokierunkowa)*,**
- ***„losowa"*, tzn. każdy indeks (adres) z przedziału [*0,m-1*] powinien być jednakowo prawdopodobny (stąd nazwa: funkcja mieszająca).**

Ze względu na fakt, że $\mathscr{K}$ >> $\mathscr{I}$, dla wszystkich wartości „dobrej" funkcji mieszającej istnieje wiele argumentów funkcji (składowych kluczowych) z nimi związanych.

O takich wartościach składowych kluczowych, które dają identyczne wartości funkcji mieszającej, mówi się, że są w kolizji.

Dygresja

Pojęcia doskonałej oraz minimalnej doskonałej funkcji mieszającej dotyczą przypadku, w którym znany jest *a priori* podzbiór składowych kluczowych, które pojawią się podczas operacji słownikowych. Jeżeli moc tego podzbioru *n* ≤ *m*, gdzie *m* – moc typu indeksującego dla tablicy mieszającej, to wtedy można przed inicjacją słownika wyznaczyć taką funkcję mieszającą, która jest różnowartościowa, a więc zapobiega kolizji.

Istnienie kolizji powoduje, że proces wyszukiwania elementu o kluczu *k* w tablicy mieszającej *A*[ ] powinien przebiegać w sposób następujący:

- w pierwszym kroku operacji wyszukiwania obliczyć związany z kluczem *k* indeks:

  $h = H(k)$,

- w drugim kroku - *co jest konieczne* - sprawdzić, czy element o kluczu *k* jest rzeczywiście wskazywany przez indeks *h* w tablicy *A*, tj. sprawdzić, czy

  $A[H(k)].klucz = k$?

- w przypadku błędnego wyznaczenia adresu obiektu, należy w większości metod implementacji tablicy mieszającej wyznaczyć indeks alternatywny *h'*, ponownie sprawdzić, czy wskazuje on poszukiwany obiekt, zaś w przypadku niepowodzenia wyznaczyć kolejny indeks *h''*, itd.

Skuteczność (czytaj: złożoność obliczeniowa) wyszukiwania elementu w tablicy mieszającej zależy od właściwego wyboru funkcji mieszającej oraz metody postępowania w przypadku stwierdzenia kolizji w fazie umieszczania nowego elementu danych w słowniku.

Dygresja

Zalecanym rozwiązaniem, nie zawsze dogodnym z innych względów, jest przyjęcie jako mocy typu indeksującego pewnej liczby pierwszej.

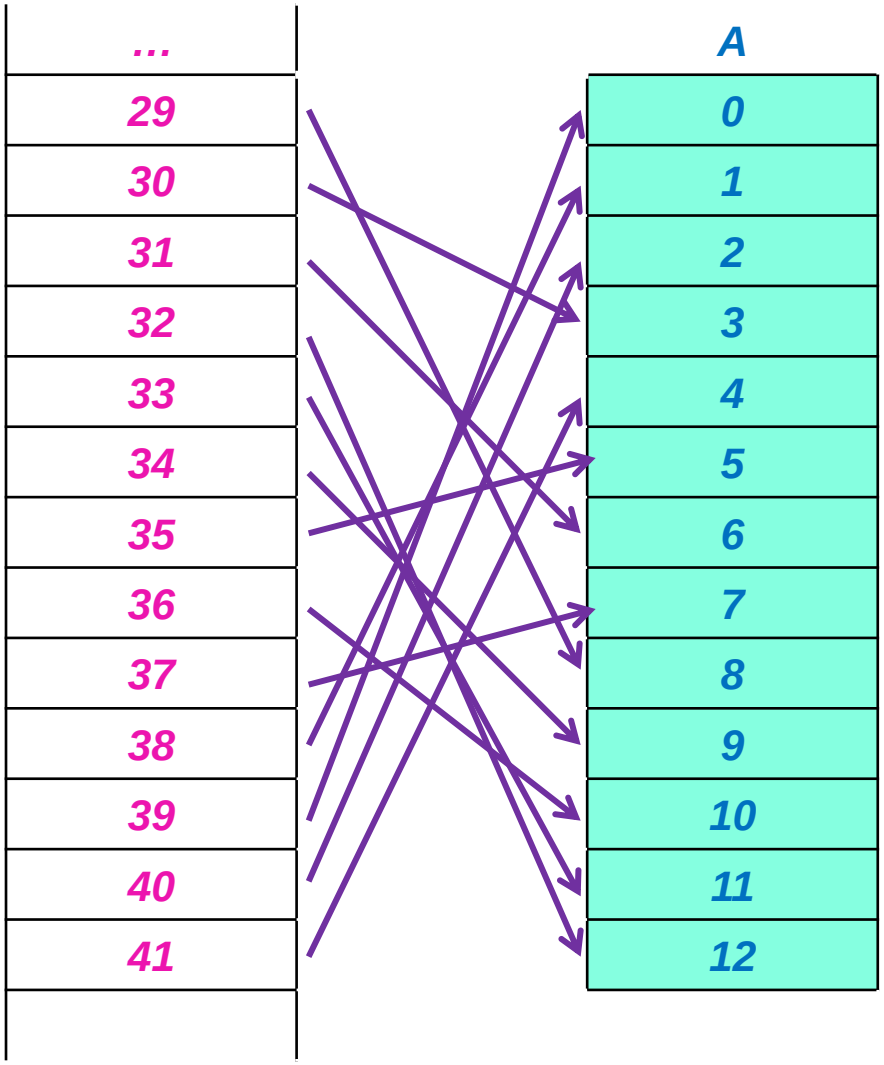
Przykład „dobrej” funkcji mieszającej

Założenie

Każda wartość składowej kluczowej *k* ma swój odpowiednik w postaci całkowitej liczby nieujemnej *K*.
Niech *m = 13* (liczba pierwsza).

Propozycja funkcji mieszającej

$$h = H(K) = \begin{cases} 0, \text{ gdy } K = 0 \text{ lub } 13 \mid K \\ \text{albo} \\ 2^{K \bmod 13} \bmod 13 \end{cases}$$



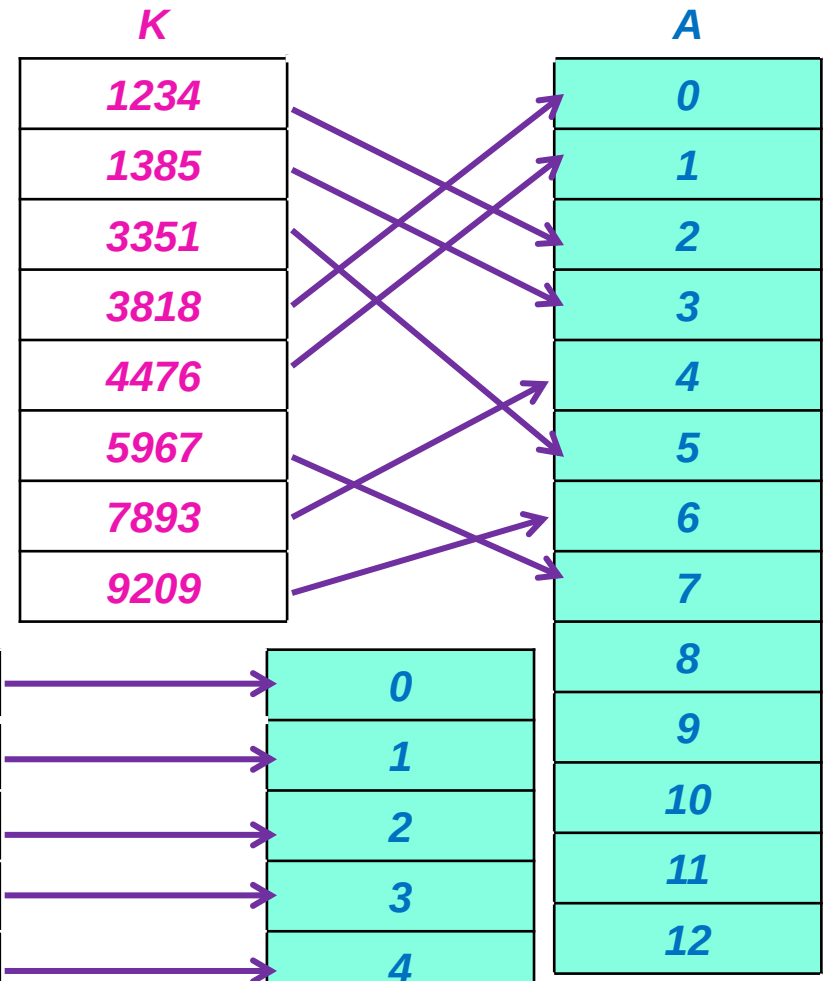
Ostrożnie z doskonałą funkcją mieszającą (?!?)

Założenie

Wiadomo, że w słowniku mogą się pojawić wyłącznie następujące klucze: 1234, 1385, 3351, 3818, 4476, 5967, 7893 i 9209.

Poszukując doskonałej funkcji mieszającej przyjęto *m = 13* (liczba pierwsza).

„Po żmudnych poszukiwaniach" sformułowano propozycję funkcji mieszającej:

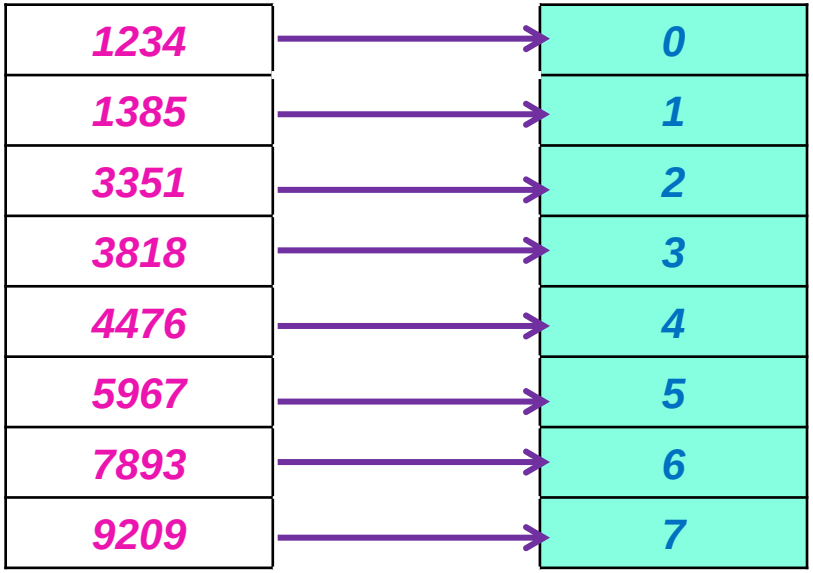


$$h = H(K) = (5K + 7) \bmod 13$$

Oszczędność pamięci ↑

Oszczędność czasu ↓

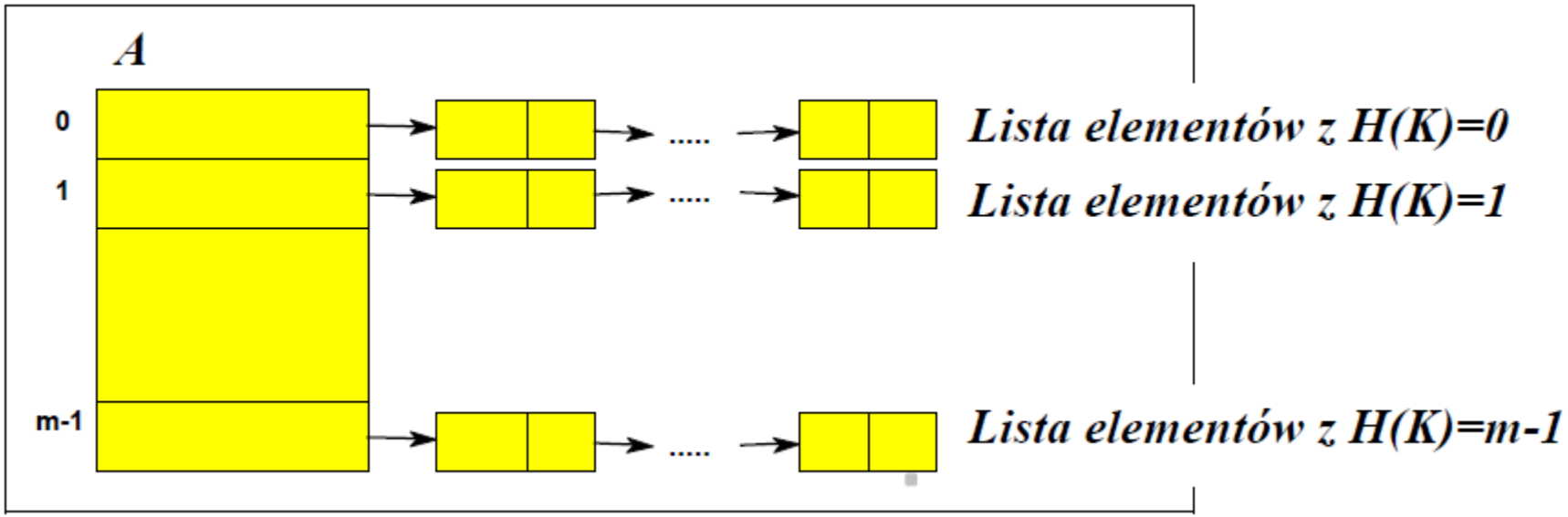
I wtedy pojawił się ktoś rozsądny, proponując „brutalne" zastosowanie rozwiązania „look-up table"…



Wstawianie nowego elementu danych do tablicy mieszającej

Metoda łańcuchowa

W metodzie łańcuchowej z każdą pozycją w tablicy jest związana jest lista, czyli *łańcuch* rekordów, które w polach przeznaczonych na dane przechowują klucze i skojarzone z nimi „breloki” (typem podstawowym dla tablicy jest referencja do listy rekordów o typie zgodnym z typem elementów danych).



Operacje słownikowe polegają na odnalezieniu listy powiązanej z elementem tablicy mieszającej *A* o indeksie *H(K)*, a następnie na wykonaniu odpowiedniej operacji na tej liście z argumentem *k*.

W przypadku pesymistycznym złożoność czasowa metody łańcuchowej jest taka sama, jak w przypadku „klasycznej" implementacji listowej, tzn. *O(n)* dla operacji *search*, *insert* i *delete*, oraz *O(m)* dla operacji *construct*.

W przypadku złożoności oczekiwanej, przy założeniu, że prawdopodobieństwo otrzymania danej wartości funkcji mieszającej jest takie samo dla każdego elementu, prawdopodobieństwo wstawienia elementu do określonej listy wynosi *1/m*.

Dla słownika, do którego wstawiono *n* elementów, oczekiwana liczba elementów trafiających na jedną listę wynosi *n/m*. Stąd złożoność oczekiwana dla operacji *search*, *insert* oraz *delete* wynosi *n/m+O(1)*.

Metoda łańcuchowa wymaga pamięci na przechowywanie dowiązań (wskaźników), stąd jej złożoność pamięciowa wynosi *m+n+O(1)*.

***Zaletami* metody łańcuchowej są: jej *prostota* i *dobra oczekiwana złożoność czasowa*, *wadą* natomiast stosunkowo *duże obciążenie pamięci*.**

Metody adresowanie otwartego

Wszystkie elementy danych przechowywane są w tablicy mieszającej (typem podstawowym dla tablicy jest typ zgodny z typem elementu danych).

W przypadku kolizji należy wyznaczyć inne wolne miejsce w tablicy mieszającej do ulokowania nowego elementu danych.

Zaproponowano wiele mniej lub bardziej wyrafinowanych strategii wyznaczania alternatywnego miejsca dla wstawianego nowego elementu danych w przypadku kolizji składowych kluczowych:

- **adresowanie liniowe**
- **adresowanie kwadratowe**
- **mieszanie podwójne**
  - **„klasyczne”**
  - **wariant „Robin Hood’a” (*Robin Hood hashing*) - 1986**
  - **wariant „kukułki” (*cuckoo hashing*) - 2001**
  - **wariant „gry w klasy” (*hopscotch hashing*) - 2008**
  - **…**

Adresowanie liniowe

W metodzie tej w przypadku kolizji, tzn. wtedy gdy miejsce pod indeksem *A[H(k)]* jest zajęte oraz *A[H(k)] ≠ k*, poszukuje się miejsca do wstawienia elementu o kluczu *k* pod kolejnymi adresami *H(k)+1*, *H(k)+2*, *... (mod m)*.

W miarę zapełniania tablicy znajdujące się w niej elementy mają tendencję do grupowania się, przez co operacje słownikowe wykonują się znacznie wolniej.

Przykład

H(k) = k mod 10

	A
0	869
1	
2	332
3	703
4	512
5	175
6	495
7	152
8	
9	239

Przykład pokazuje, że puste miejsca znajdujące się za wypełnioną grupą „komórek”, mają znacznie większa szansę na to, że zostaną zapełnione, niż pozostałe.

Prawdopodobieństwo „zajęcia” takiej „komórki” wynosi:

***(rozmiar grupy + 1)/m*.**

Prawdopodobieństwo „zajęcia” dla „samotnych komórek” wynosi tylko:

***1/m*.**

Raz utworzone zgrupowanie ma skłonność do dalszego wzrostu, a im staje się większe, tym większe prawdopodobieństwo, że jeszcze się powiększy. Wpływa to ujemnie na efektywność tablic mieszających przy zapisywaniu i odczytywaniu danych.

Mieszanie podwójne („klasyczne”)

Zamiast przyrostu *1* stosowanego do wyznaczenia wolnego miejsca w tablicy, przyjmuje się przyrost określony przez drugą funkcję mieszającą *H’(k)*.
Funkcja ta powinna spełniać następujące warunki:

- ***H’(k) > 0***
- ***H’(k)* jest względnie pierwsze z *m* (najlepiej, gdy *m* jest liczbą pierwszą)**
- ***H’(k)* jest „istotnie różna” od *H(k)***

Pierwsza funkcja mieszająca *H(k)* wykorzystywana jest do *wyznaczania pierwotnego adresu elementu danych*, zaś druga - *H’(k)* - do *rozwiązania problemu kolizji*.

Ciągiem sprawdzanych pozycji (indeksów, adresów) jest teraz ciąg:

***H(k)*, *H(k)+H’(k)*, *...*, *H(k)+i* H’(k)*, *...* (wszystkie adresy *modulo m*).**

Rozmiar tablicy *m* powinien (choć nie musi) być liczbą pierwszą z tego względu, by każda możliwa wartość indeksu mogła się w tym ciągu pojawić.

Eksperymenty pokazują, że problem grupowania praktycznie znika, ponieważ ciąg pozycji zależy od wartości *H'(k)*, które z kolei zależą od klucza *k*.

Przykład

H(k) = k mod 10

H'(k) = ((8 – k) mod 10) mod 7

if(H'(k)=0) H'(k) = 1

	A
0	
1	
2	332
3	703
4	
5	175
6	
7	
8	495
9	239

Oczekiwana złożoność czasowa dla różnych metod mieszania

($\alpha = n/m$ - współczynnik wypełnienia tablicy mieszania)

	Liczba prób dla search(k,S), $k \in S$ (sukces)	*Liczba prób dla search(k,S), $k \notin S$ (porażka)*
Metoda łańcuchowa	$1+\frac{\alpha}{2}$	$1+\alpha$
Adresowanie liniowe	$\frac{1}{2}+\frac{1}{2(1-\alpha)}$	$\frac{1}{2}+\frac{1}{2(1-\alpha)^2}$
Mieszanie podwójne	$-\frac{1}{\alpha}\ln(1-\alpha)$	$\frac{1}{1-\alpha}$

Średnia liczba sprawdzeń przy wyszukiwaniach zakończonych sukcesem albo porażką dla różnych metod rozwiązywania kolizji

α	Metoda łańcuchowa		Adresowanie liniowe		Mieszanie podwójne	
	Sukces	*Porażka*	*Sukces*	*Porażka*	*Sukces*	*Porażka*
0.05	1.0	1.1	1.0	1.1	1.0	1.1
0.20	1.1	1.2	1.1	1.3	1.0	1.2
0.40	1.2	1.4	1.3	1.9	1.3	1.7
0.60	1.3	1.6	1.8	3.6	1.5	2.5
0.80	1.4	1.8	3.0	13.0	2.0	5.0
0.95	1.5	2.0	10.5	200.5	3.2	20.0

Cuckoo hashing

Jeśli pozycja nowego elementu określona przez *H(k)* jest zajęta, to nowy element „wyrzuca” z niej dotychczasowego „lokatora” zajmując tą pozycją, a „wyrzucony” element jest umieszczany na pozycji wynikającej z wartości *H’(k)*(„wyrzucając” z niej ewentualnego dotychczasowego „lokatora”, itd., aż do znalezienia wolnej pozycji).

Zaleta - wyszukiwanie elementu jest klasy O(1).

Wada – „wolne” wstawianie po przekroczeniu 50% wypełnienia tablicy.

Przykład

H(k) = k mod 10

H’(k) = ((k + 7) mod 13) mod 10

Proces ten może się zapętlić, np. wtedy, gdy:

H(k1) = H(k2) i H’(k1) = H’(k2)

lub

H(k1) = H’(k2) i H’(k1) = H(k2)

Wtedy rozmieszcza się elementy w tablicy zgodnie z nową funkcją(-ami) mieszającą(-ymi).

	A
0	175
1	332
2	512
3	703
4	
5	869
6	
7	
8	495
9	239

Metoda połączonych łańcuchów (*ang.: coalesced hashing*)

Metoda integruje koncepcję adresowania liniowego ze strukturami listowymi, przy czym lista jest tworzona wewnątrz tablicy mieszającej , zaś rolę dowiązania do następnika spełnia dodatkowa składowa rekordu skojarzonego z pojedynczym elementem tablicy mieszającej.

Dla klucza będącego w kolizji z innym szuka się pierwszej wolnej pozycji, a jej indeks umieszcza się przy kluczu znajdującym się już w tablicy.

Ponadto dla elementów danych, na które nie ma już miejsca w tablicy, można przeznaczyć tzw. *obszar nadmiarowy* przydzielany dynamicznie (np. *klasyczną listę*).

Przykład

H(k) = k mod 10

A	*key*	*next*
0	869	NA
1		NA
2	332	4
3	703	NA
4	512	7
5	175	6
6	495	NA
7	152	NA
8		NA
9	239	0

Metoda adresowania kubełkowego

W tej metodzie kolizja jest rozwiązywana przez umieszczenie kolidujących elementów danych na tej samej pozycji w tablicy. W tym celu z każdym adresem w tablicy trzeba związać nie miejsce na pojedynczy element danych, ale *kubełek* (*ang.: bucket*), czyli blok pamięci mogący pomieścić wiele elementów danych.

Zastosowanie kubełków nie pozwala całkowicie uniknąć problemu kolizji. Może się bowiem zdarzyć, że zadeklarowana statycznie *„głębokość kubełka”* okaże się niewystarczająca – *kubełek zostanie zapełniony*.

W takim przypadku nowy element danych należy umieścić w innym kubełku, wyznaczonym zgodnie z regułami adresowania otwartego.

Przykład

H(k) = k mod 10

głębokość kubełka = 2

adresowanie liniowe

	A	
0		
1		
2	332	512
3	703	152
4		
5	175	495
6		
7		
8		
9	239	869

Innym rozwiązaniem stosowanym w przypadku *przepełnienia kubełków* jest umieszczanie kolejnych elementów o tej samej wartości funkcji mieszającej w tzw. *obszarze nadmiarowym*.

Każdy kubełek zawiera wówczas pole informacyjne (*znacznik*), wskazujące na to, czy w danym obszarze należy kontynuować poszukiwanie, czy też nie.

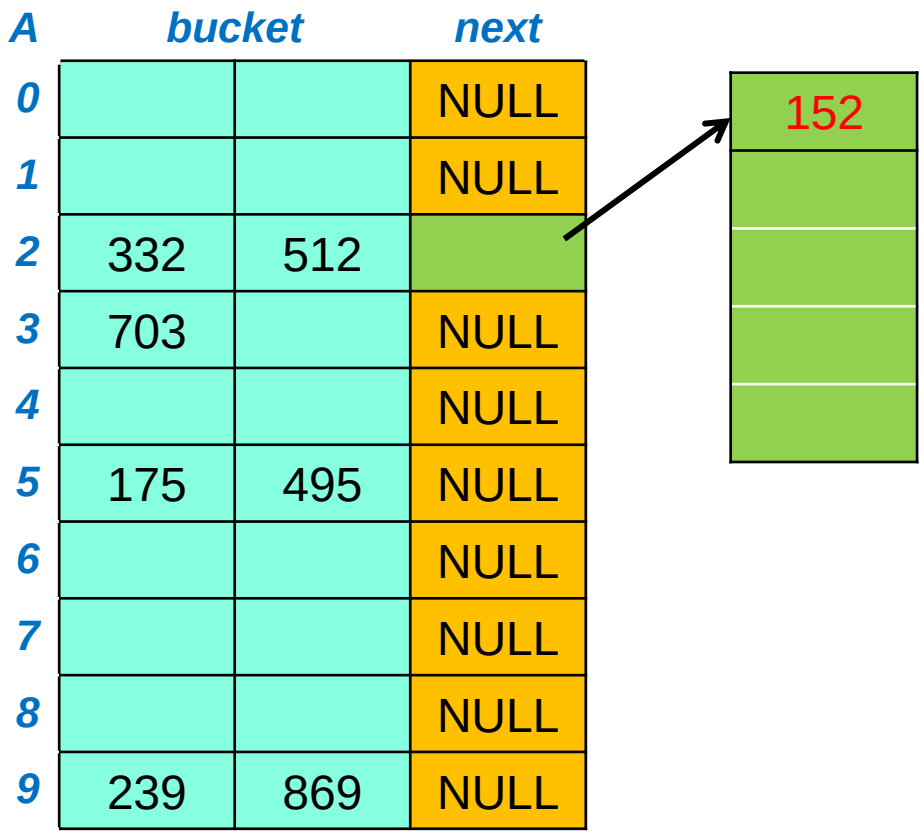
Organizacja obszaru nadmiarowego może przyjąć np. formę typową dla *metody łańcuchowej* – wówczas znacznik ten może zawierać *wskaźnik do początku listy* związanej z danym kubełkiem w obszarze nadmiarowym.

Przykład

H(k) = k mod 10

głębokość kubełka = 2

obszar nadmiarowy – lista (łańcuch)



Usuwanie elementu danych z tablicy mieszającej

Metoda łańcuchowa

W przypadku metody łańcuchowej problem usunięcia elementu danych z tablicy mieszającej sprowadza się do zagadnienia *usunięcia elementu z listy liniowej* skojarzonej z indeksem tablicy zgodnym z wartością funkcji mieszającej klucza usuwanego elementu (najpierw jednak trzeba go na tej liście odnaleźć).

Metody adresowania otwartego

Dla metod adresowania otwartego zagadnienie usunięcia elementu danych z tablicy mieszającej jest już bardziej kłopotliwe.

Załóżmy, że podczas wstawiania elementy danych o identycznych wartościach funkcji mieszającej dla ich kluczy pojawiały się w kolejności:

$$E(k_1), E(k_2), \ldots, E(k_i), E(k_{i+1}), \ldots, E(k_m)$$

i były lokowane w tablicy zgodnie z przyjętą metodą wyszukiwania wolnego miejsca w przypadku kolizji (adresowanie liniowe, mieszanie podwójne, itp.).

Po usunięciu elementu o kluczu k_i *traci się dostęp* do elementów o kluczach $k_{i+1}, \ldots, k_m$.

Sposobem na ominięcie tego problemu jest zaznaczenie w tablicy mieszania faktu usunięcia elementu danych o określonej wartości klucza *bez usuwania tej wartości* z miejsca poprzednio zajmowanego. Miejsce to może zająć w przyszłości *kolejny wstawiany element* o identycznej wartości funkcji mieszającej klucza.

Nie jest to rozwiązanie korzystne w aplikacjach, w których operacje usuwania elementów ze słownika są częste, gdyż prowadzi do *istotnego wydłużenia czasu operacji wyszukiwania*.

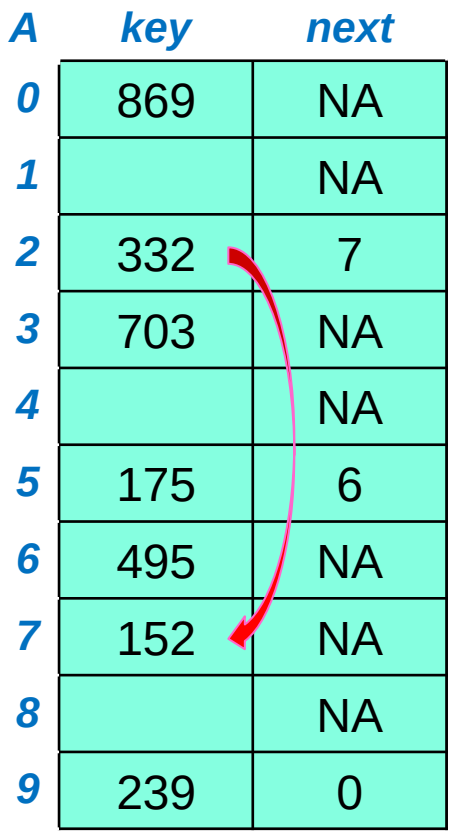
Mechanizmem pomocniczym, przywoływanym po pewnej liczbie operacji usunięcia elementu danych z tablicy, jest procedura *przepisywania* w miejsca zajęte przez elementy oznaczone jako usunięte tych elementów, które do tablicy zostały wstawione później, oraz zaznaczania zwolnionych przez nie indeksów jako ponownie dostępnych do wykorzystania (tzw. *przeindeksowanie tablicy mieszającej*).

Metoda połączonych łańcuchów

Dzięki dowiązaniom usunięcie wskazanego elementu danych po jego odnalezieniu sprowadza się do odpowiedniej modyfikacji informacji o następniku w elemencie danych poprzedzającym na „liście” zbudowanej z dowiązań element usuwany.

Przykład

Usunięcie elementu o kluczu 512



Koniec części 9