

# Хештаблица

Лекция 9 по СДА, Софтуерно Инженерство  
Зимен семестър 2018-2019г  
д-р Милен Чечев

# Какво научихме до сега

[illegible][illegible]

implementation	guarantee			average case			ordered ops?	key interface
	search	insert	delete	search	insert	delete		
sequential search (unordered list)	$n$	$n$	$n$	$n$	$n$	$n$		equals()
binary search (ordered array)	$\log n$	$n$	$n$	$\log n$	$n$	$n$	✓	compareTo()
BST	$n$	$n$	$n$	$\log n$	$\log n$	$\sqrt{n}$	✓	compareTo()
AVL BST	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$	✓	compareTo()
hashing	$n$	$n$	$n$	$1^\dagger$	$1^\dagger$	$1^\dagger$		equals() hashCode()

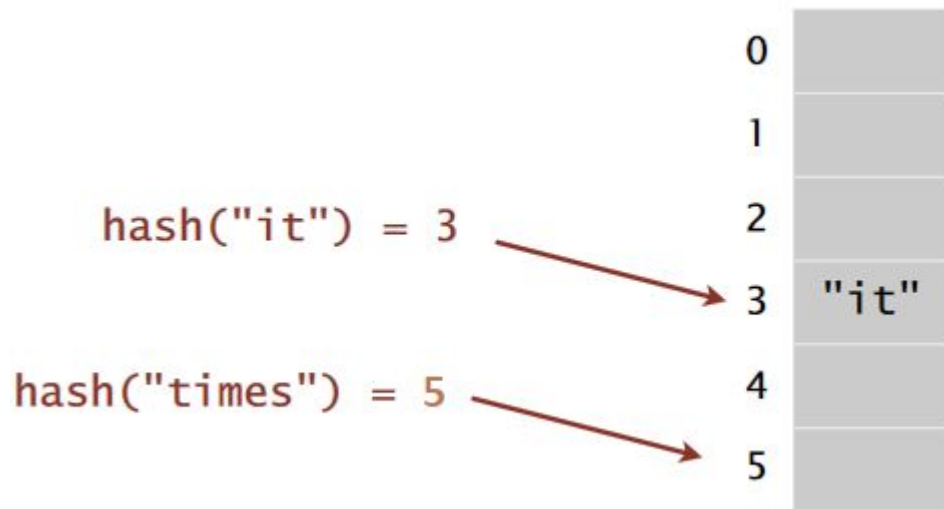
† under suitable technical assumptions

# Хеш функция

- Функция, която преобразува обект в число в определени граници
- Свойства:
  - Хеш функцията трябва да е бърза.
  - Трябва да връща винаги един и същи резултат за един и същ обект

# Хеш множество

Ненаредена структура от данни, която може да запомня обекти, като позволява търсенето за това дали обект е вече добавен да става с сложност  $O(1)$  в средният случай.



# Хеш таблица

Структура от данни, която съдържа двойки (ключ, стойност), която позволява добавяне и изваждане на нови двойки и търсене по ключ със сложности  $O(1)$  в средният случай.

	0	1	2	3	4	5	6	7
keys[]		E	S			R	A	
vals[]		1	0			3	2	

# Хеш таблица

Основна идея:

Ако имаме хешираща функция, която да хешира ключовете в интервала  $0..K$  бихме могли да използваме масив от  $K$  елемента за запомняне на стойностите, като от хеширащата функция ще разберем в коя клетка да пишем.

	0	1	2	3	4	5	6	7
keys[]		E	S			R	A	
vals[]		1	0			3	2	

# Основен проблем - Колизии

- Колизии - два обекта имат една и съща хеш стойност  $h(k_1)=h(k_2)$ 
  - Двата обекта са различни, но поради ограничението в размера на пространството за хеширане, техните стойности съвпадат.
  - Пример за колко лесно се срещат колизии - Използваме универсална хешираща функция, която хешира студентите от една група спрямо тяхната рождена дата(ден и месец) - Ако имаме 23 студента в групата то вероятността някой двама от тях да са родени на една дата е над 50%([Birthday problem](#))



# Какво правим ако има колизия?

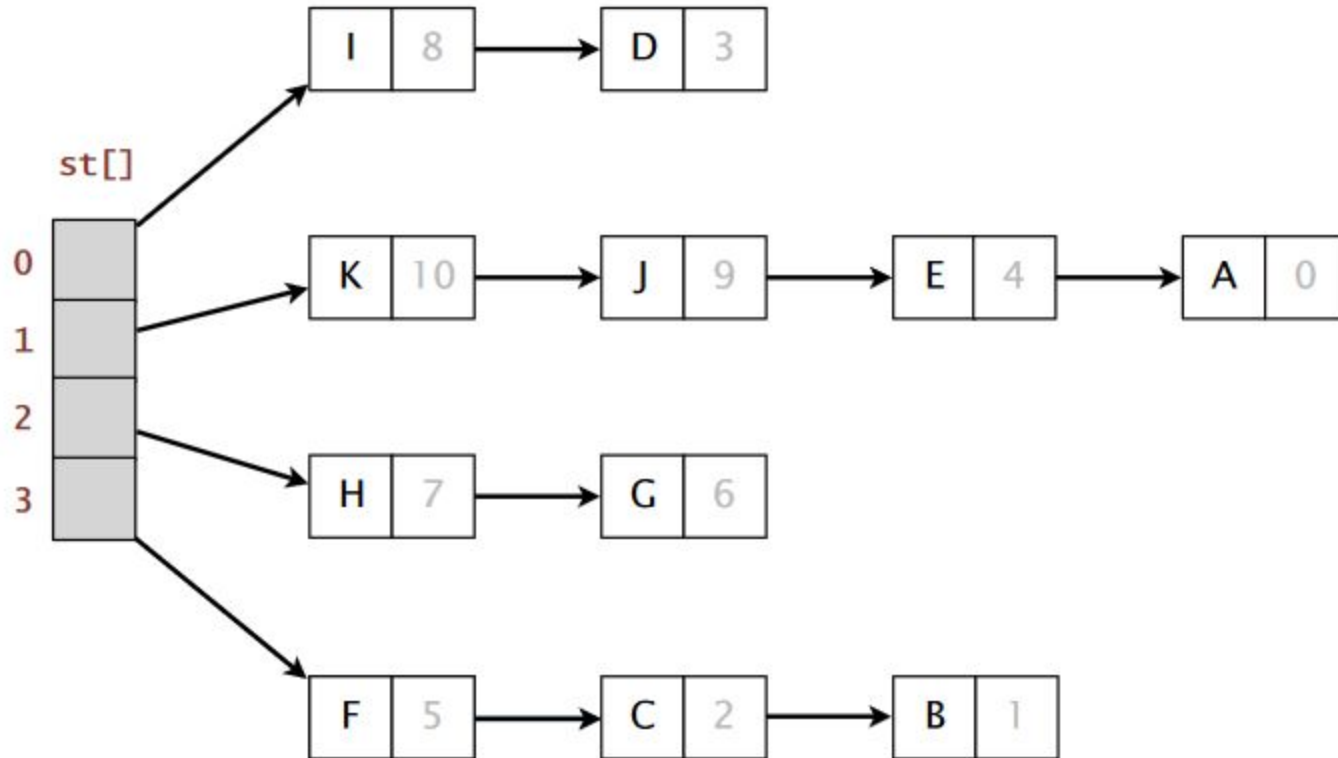
основни стратегии:

- Запазване на няколко стойности в една клетка
- Използване на следваща празна клетка
- Използване на няколко хеширащи функции

# Запазване на няколко стойности в една клетка (separate chaining)

Основна идея: Вместо една стойност пазим списък със всички добавени двойки. Като при търсене търсим в списъка.

# Separate chaining hash table



# Separate chaining hash table

Insert - изчисляване на хеш функцията, която връща число от 0 до  $n$ , добавяне в списъка, който е на позиция  $n$

Търсене - изчисляване на хеш функцията, която връща число от 0 до  $n$ , търсене в списъка, който е на позиция  $n$

При добра хешираща функция и броя на числата ако са около  $\frac{1}{4}$  от големината на хеш таблицата - константна сложност(в средният случай)

Сложността в най-лошият случай:  $O(n)$  - ако всички добавени елементи са с един и същ хеш код и всички числа са застанали в списък

# Увеличаване на големината на таблицата

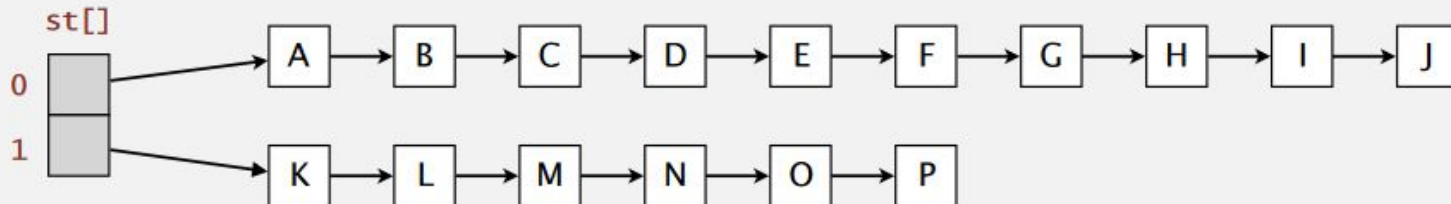
Цел: Търси се съотношението на броя на елементите към броя на клетките в таблицата да е константа.

Примерна стратегия:

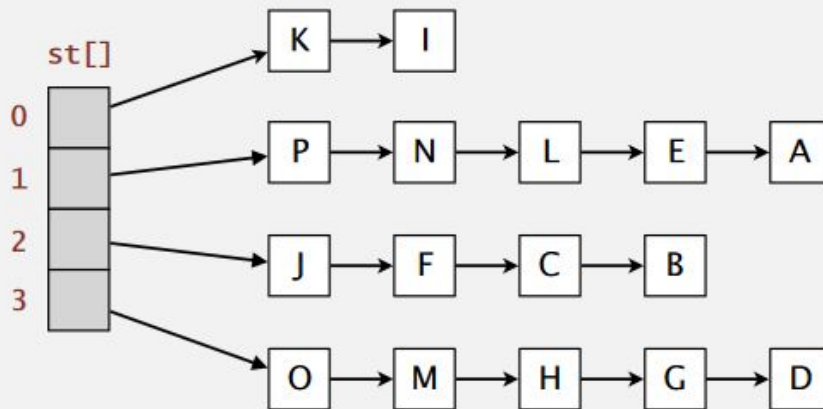
- Удвояваме големината на таблицата ако броят на елементите стане 8 пъти по голям от броя на клетките в таблицата
- Намаляме големината на таблицата на две ако броя на елементите стане само 2 пъти по-малък от броя на клетките в таблицата.
- След промяна на големината на таблицата всички елементи се пре хешират!

# Увеличаване на големината на таблицата

before resizing ( $n/m = 8$ )



after resizing ( $n/m = 4$ )



# Търсене на следваща празна клетка (Linear probing)

- Основна идея ако клетката е заета търсим следваща свободна клетка и разполагаме двойката там(ключа и стойността)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
keys[]	P	M			A	C		H	L		E				R	X
vals[]	11	10			9	5		6	12		13				4	8

# Linear Probing (проблеми)

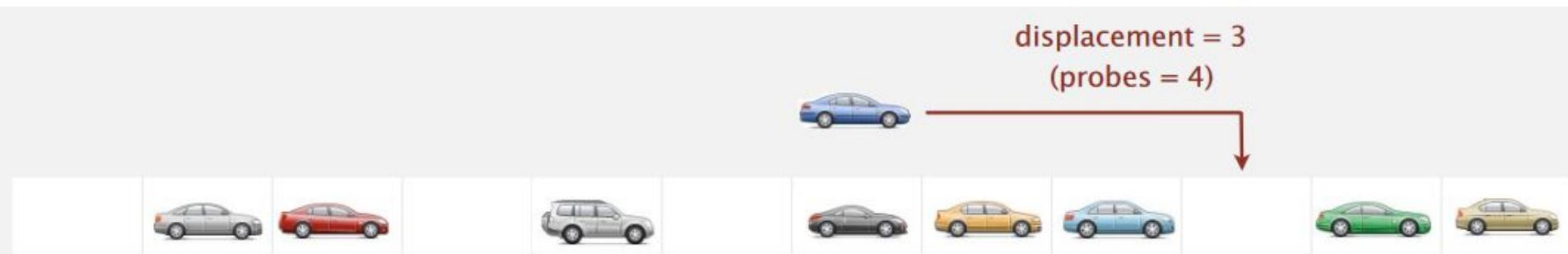
Clustering - при образуване на групи от елементи в масива с ключове става все по вероятно някоя от стойностите да бъде получена като хеш от нов елемент и оттам да се търси следваща стойност (което пък от своя страна разширява и групата)



# Проблема за паркиране на Кнут

$N$  коли искат да паркират на паркинг с  $M$  места, като всяка от тях иска да паркира на свое си паркомясто  $i$  или на следващо такова.

Въпрос: Какво е средното отклонение от желаното място за паркиране



Отговор: ако  $M=N/2$  то  $\sim 1/2$

Ако  $M=N$  то  $\sim \sqrt{\pi n/8}$  Извод: Не може да се позволи да е прекалено пълна таблицата

# Хеш таблица с Linear Probing

За да е оптимално търсенето - трябва да се поддържа  $N/M < 1/2$

За това следваме следната процедура:

- Ако  $N/M > 1/2$  удвояваме големината на таблицата
- Ако  $N/M < 1/8$  намаляваме на половина големината на таблицата

При намаляване или увеличаване на големината на таблицата за всички елементи до момента се преизчислява хеш функцията

# Изтриване в хеш таблица с Linear Probing

- Не може просто да изтрием ключ, която е между други ключове понеже ще направи дупка и след това ще провали евентуално търсене на ключ.

Решение: Вместо изтриване отбелязваме за изтрито.

**before deleting S**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
keys[]	P	M			A	C	S	H	L		E				R	X
vals[]	10	9			8	4	0	5	11		12				3	7

**after deleting S ?**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
keys[]	P	M			A	C		H	L		E				R	X
vals[]	10	9			8	4		5	11		12				3	7

doesn't work, e.g., if  $\text{hash}(H) = 4$



# Двойно хеширане

Основна идея: Имаме две(или повече) хеш функции и когато се получи колизия със първата се изчислява хеш от втората хеш функция и се търси да се добави стойността на място  $\text{хеш1} + \text{хеш2}$ , ако и то е заето се търси място на  $\text{хеш1} + 2 * \text{хеш2}$  и т.н. Докато се намери място.

# Известни Хеш функции

Известни криптографски хеширащи алгоритми за хеширане на пароли:

MD4, MD5, SHA-0, SHA-1, SHA-2, SHA-256, SHA-512

Дали горните хеш алгоритми са подходящи за хеш таблица?

**Не! Прекалено бавни са.** (повече от 600ms средно време за хеширане на 36 букви)

# Каква хеш функция да използваме?

Цел: стойностите, които хешираме да се разположат максимално еднакво в целият интервал.

Може да ползваме различни специфични хеш функции за нашият конкретен случай, като ако нямаме конкретна функция в предвид, то може да ползваме универсална хешираща функция.

Универсално хеширане:

$$h(x) = ((a \cdot x + b) \bmod p) \bmod n$$

Където  $n$  е големината на таблицата.  $p$  е просто число по-голямо от  $n$

$a$  и  $b$  са произволни числа по-малки от  $p$

# Как да хешираме масиви?

“ $31x + y$  rule”:

Обхождаме масива и последователно акумулираме стойностите му като умножаваме сумата до сега с 31 и добавяме новото число.

Защо 31?

- Просто, нечетно число.
- $31 * i == (i \ll 5) - i$  (лесно за изчисление)

# Как да хешираме разнообразни типове данни?

Double - може да го разгледаме като два int

Long - и той може да се разглежда като два int

String - масив от char

Обект - множество от член данните си(всяка от която вече знаем как да хешираме). Множеството може да го разгледаме като масив от разнородни елементи.



# Обобщение

Хеш таблиците и балансираните дървета са доста оптимални структури за съхранение и търсене.

При балансираните дървета имаме гарантирана сложност в най-лошият случай от  $O(\log(n))$  за основните операции(добавяне, изтриване, търсене) докато при хеш таблицата нямаме такава гаранция за сложност в най-лошият случай, но практически работи в доста реални случаи работи по-добре и по-бързо от балансираните дървета(в средният случай).

# Какво следва?

Граф алгоритми!

