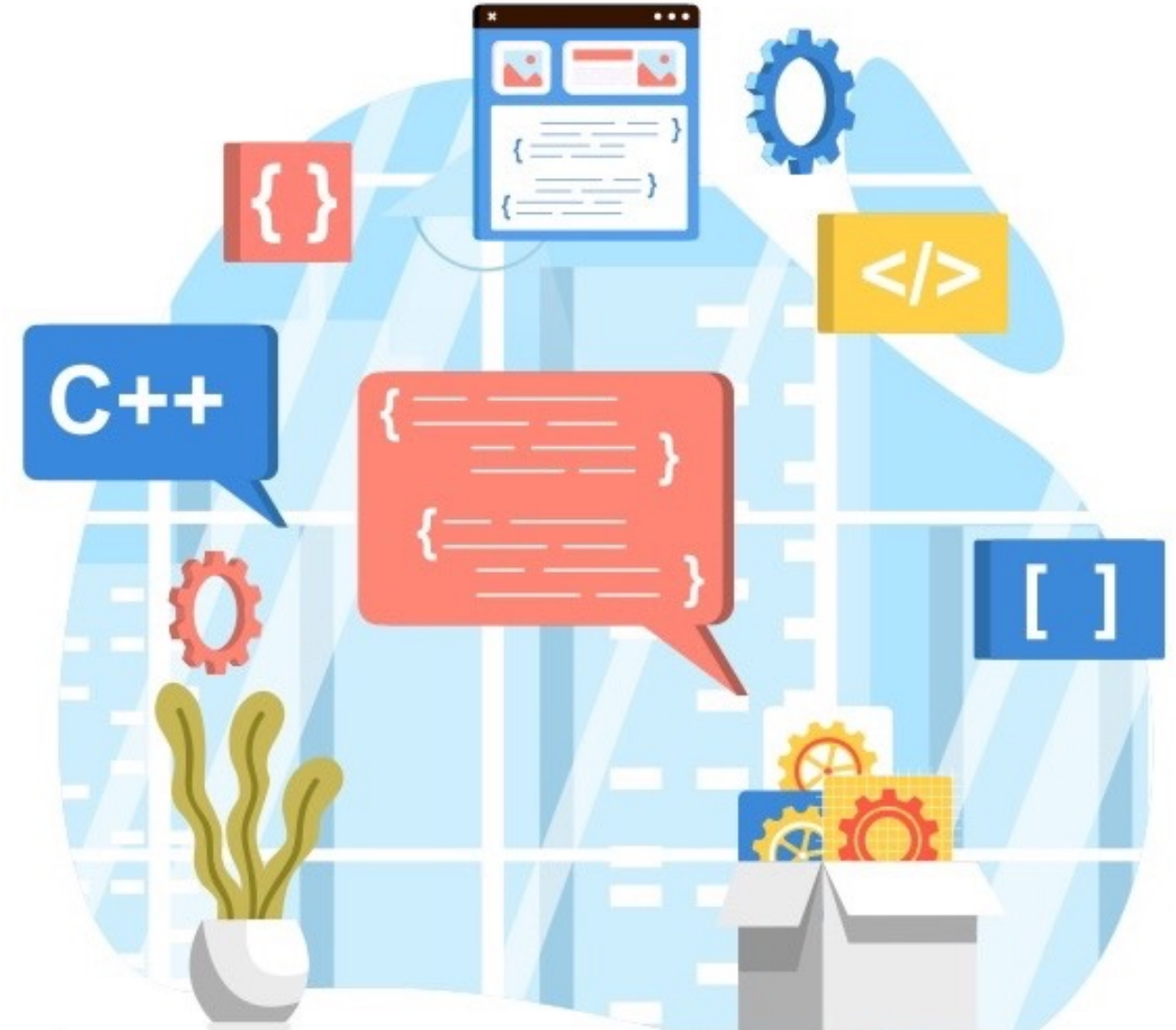
BẢNG BĂM



VŨ NGỌC THANH SANG

KHOA CÔNG NGHỆ THÔNG TIN

ĐẠI HỌC SÀI GÒN

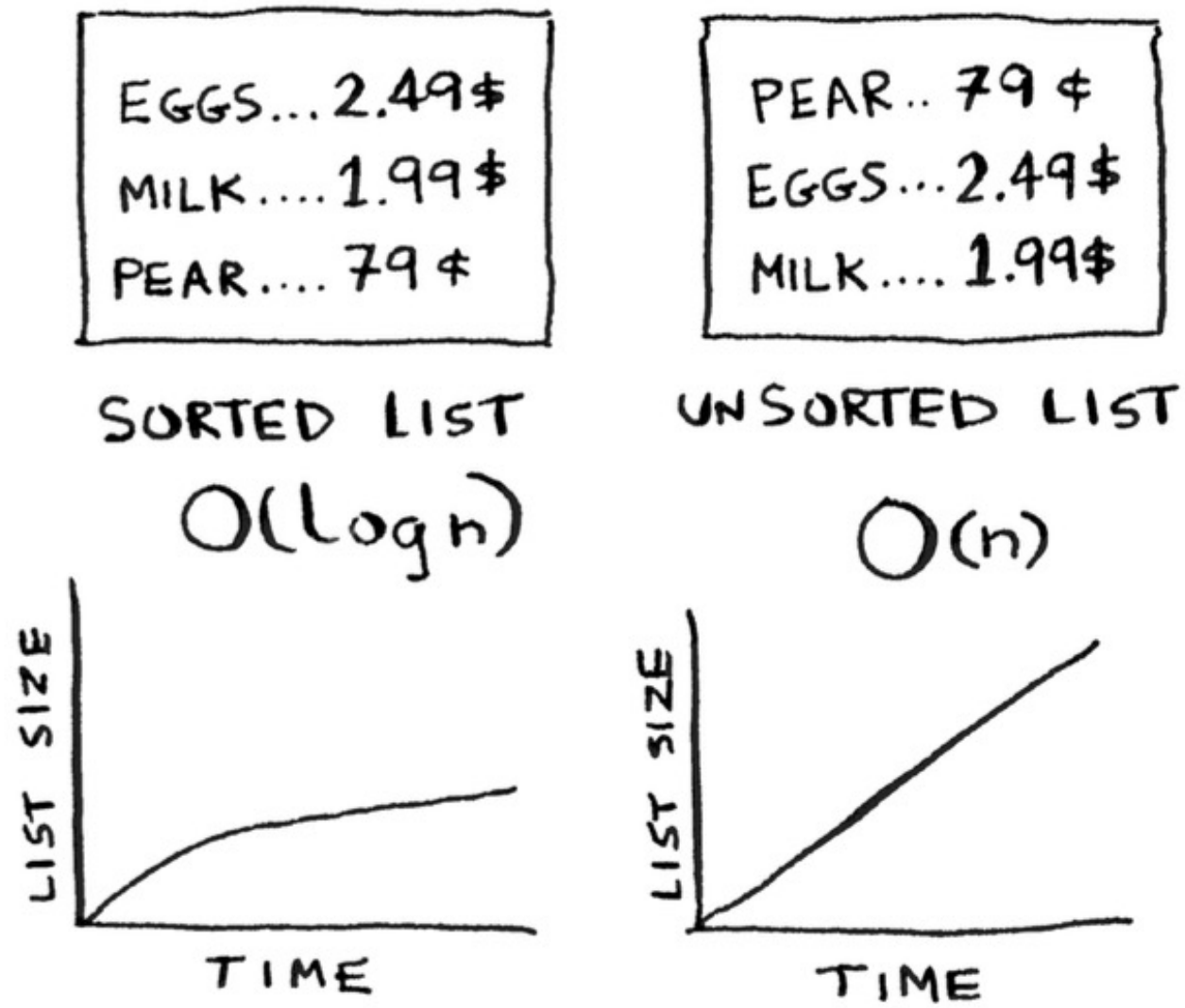
I – Introduction

- Trong các nội dung trước, quy trình chính cho việc tìm kiếm chính là thực hiện so sánh các phần tử
- Nếu mảng chưa được sắp xếp, thực hiện tìm kiếm tuyến tính,

  → Thời gian là O(n).
- Nếu mảng đã được sắp xếp, thực hiện tìm kiếm nhị phân

  → Thời gian là O(logn)
- Vậy có phương án nào thực hiện việc tìm kiếm nhanh hơn không?

I – Introduction



I – Introduction

- Để giảm độ phức tạp của thuật toán, ta sử dụng thêm bộ nhớ để chứa các giá trị cần tìm kiếm
- Ta có mảng A như sau

0	1	2	3	4	5	6
9	5	3	1	7	8	2

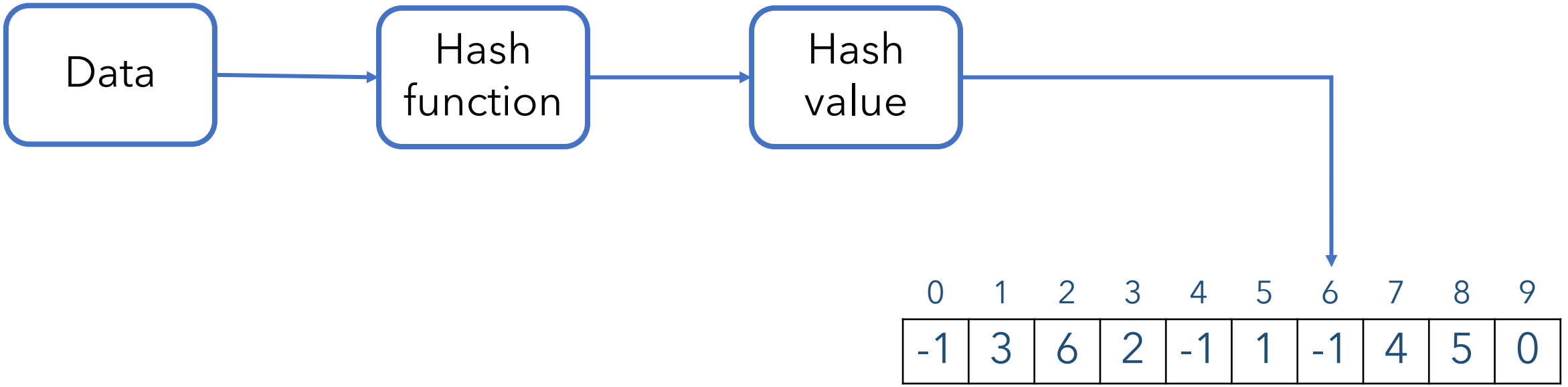
- Ta có mảng H[i] = index của giá trị i trong mảng A[]

0	1	2	3	4	5	6	7	8	9
-1	3	6	2	-1	1	-1	4	5	0

- Để xác định phần tử 7 có trong mảng A hay không chỉ cần tra H[7]

  → Độ phức tạp thuật toán O(1)

I – Introduction



I – Introduction

- Việc hashing được thực hiện qua 2 bước:
  1. Một phần tử sẽ được chuyển đổi thành 1 số nguyên bằng việc sử dụng hàm băm. Giá trị này được sử dụng như một chỉ mục (index) để lưu trữ phần tử gốc và nó sẽ được đưa vào bảng băm.
  2. Phần tử sẽ được lưu giữ trong bảng băm, nó có thể được truy xuất nhanh bằng khóa băm:
     - hash = hashfunc(key)
     - index = hash % array_size

I – Introduction

- Để sử dụng ý tưởng của bảng băm h hiệu quả, ta cần phải xác định kích thước bảng ngay từ đầu.
- Một ứng dụng cần chứa tên các biến, mỗi biến gồm tối đa 31 ký tự. Ta có thể thiết kế bảng h với kích thước bằng tổng của các ký tự trong bảng ASCII
- Ví dụ một biến có 31 ký tự toàn "z" ta có 31*122 = 3782 vị trí

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

I – Introduction

- Tuy nhiên, trong trường hợp trên, hàm h không trả về giá trị duy nhất

$$h(\text{“abc”}) = 97 + 98 + 99 = 294$$

$$h(\text{“acb”}) = 97 + 99 + 98 = 294$$

- Trường hợp này gọi là va chạm (collision), được sử dụng để đánh giá sự hữu dụng của hàm h.
- Để hạn chế sự va chạm, ta cần thiết kế hàm h phức tạm, tuy nhiên cũng cần cân bằng giữa độ phức tạp và tốc độ.

II – Hash Function

- Hàm băm được sử dụng để ánh xạ tập dữ liệu có kích thước n vào bảng băm có kích thước m.
- Hàm băm cần phải dễ tính toán và phân bố đồng đều kết quả trên bảng băm.
- Hàm băm được xem là hoàn hảo khi có thể liên kết mảng S có n phần tử vào bảng băm mà không xảy ra va chạm.

II – Hash Function

Cách thiết kế hàm băm phổ biến

- Chia lấy dư (Arimetic Modular): thực hiện phép chia lấy dư với giá trị khóa để tìm ra index trong hash table
- Chặt cụt (Truncation): Sử dụng một phần của giá trị khóa để làm index trong hash table
- Gấp (Folding): Chia giá trị khóa ra thành nhiều thành phần và mỗi thành được xử lý riêng và sau đó được kết hợp lại để xác định index trong bảng băm

II – Hash Function

- Lưu giữ các string sau đây trong bảng băm: {“abcdef”, “bcdefa”, “cdefab” , “defabc” }.
- Thiết kế hàm băm, lấy 599 chia lấy dư cho tổng các giá trị ASCII của các ký tự trong dãy.
- Các giá trị ASCII của a, b, c, d, e và f lần lượt là 97, 98, 99, 100, 101 và 102. vậy tổng của các chuỗi là 597.
- Kết quả của phép chia lấy dư 599 cho 597 là 2

II - Hash Function

Index				
0				
1				
2	abcdef	bcdefa	cdefab	defabc
3				
4				
-				
-				
-				
-				

Trong trường hợp này vẫn cần O(n) để tìm kiếm một chuỗi cụ thể

II – Hash Function

- Thiết kế lại hàm băm, tổng các mã ASCII của từng ký tự theo sau là vị trí của ký tự đó trong chuỗi. Sau đó đem kết qua chia lấy dư cho số nguyên tố 2069.

String	Hash function	Index
abcdef	(*97*1 + *98*2 + *99*3 + *100*4 + 1015 + *102*6)%2069	38
bcdefa	(*98*1 + *99*2 + 100*3* + *101*4 + 1025 + *97*6)%2069	23
cdefab	(*99*1 + *100*2 + 1013 + *102*4 + *97*5 + *98*6)%2069	14
defabc	(100*1* + *101*2 + 1023 + *97*4 + 985 + *99*6)%2069	11

II - Hash Function

Index	
0	
1	
-	
-	
-	
11	defabc
12	
13	
14	cdefab
-	
-	
-	
-	
23	bcdefa
-	
-	
-	
38	abcdef
-	
-	

III – Collision

- Giả sử có m phần tử cần được chứa vào n vị trí trong bảng

Chèn phần tử thứ	Xác suất **không** xảy ra va chạm
1	$\frac{n}{n}$
2	$\frac{n-1}{n}$
3	$\frac{n-2}{n}$
…	…
m	$\frac{n-(m-1)}{n}$

III - Collision

- Xác suất **không có va chạm** sau khi chèn m phần tử

$$\frac{(n-1)!}{(n-m)! \times n^{m-1}}$$

- Xác suất của việc **xảy ra va chạm** khi chèn m phần tử

$$1 - \frac{(n-1)!}{(n-m)! \times n^{m-1}}$$

- Việc xảy ra va chạm là không thể tránh khỏi trong thực tế

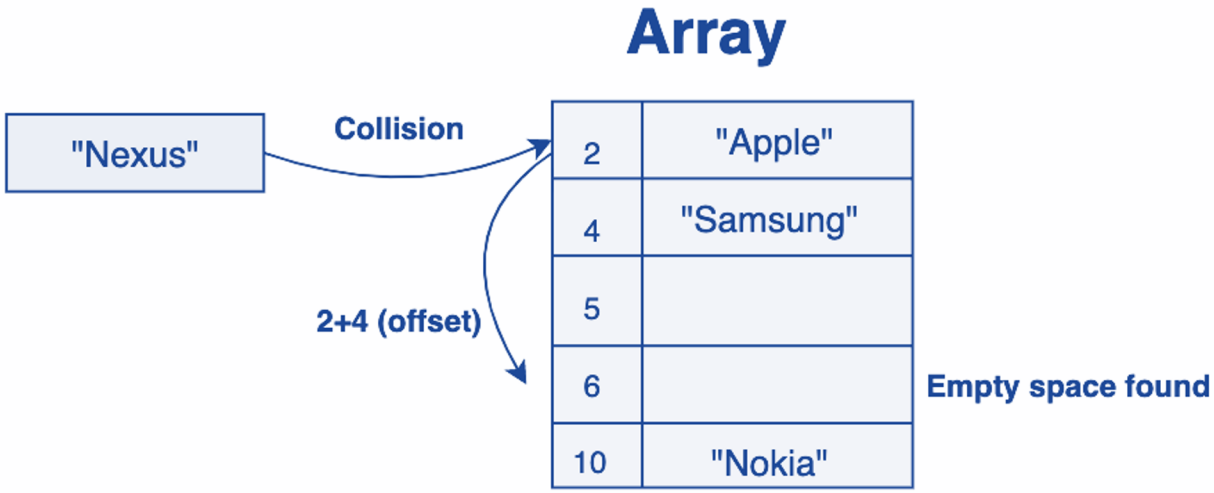
III – Collision

Kỹ thuật xử lý va chạm - **Dò tuyến tính (Linear Probing)**

Bỏ qua một chỉ mục đã được điền bằng cách thêm một giá trị offset vào chỉ mục nếu chỉ mục đó đã được điền

Key	Hash	Value
2	2%20 = 2	"Apple"
4	4%20 = 4	"Samsung"
30	30%20 = 10	"Nokia"
42	42%20 = 2	"Nexus"

Same Hash



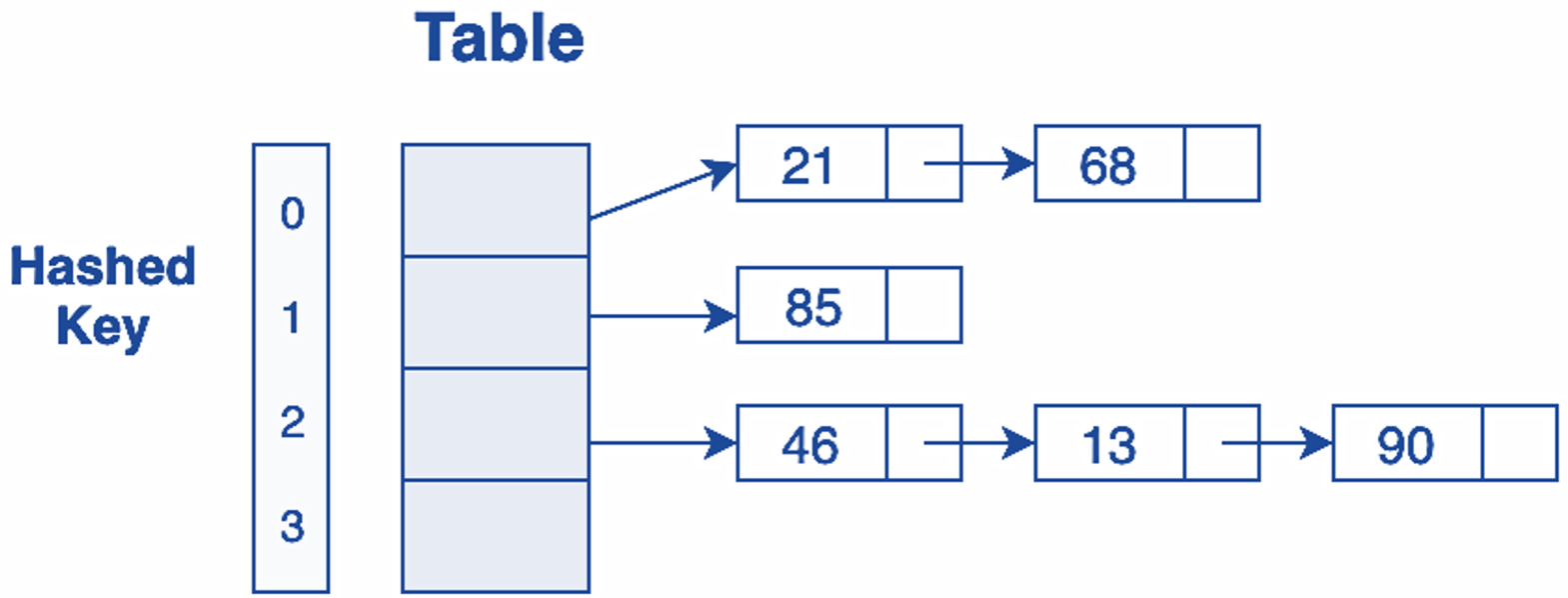
III – Collision

Kỹ thuật xử lý va chạm – **Các phương pháp kết nối (Chaining)**

- Mỗi vị trí của bảng băm chứa một con trỏ trỏ đến CTDL khác như DSLK hoặc cây. Mọi giá trị được nhập vào tại index đó sẽ được chèn vào CTDL tương ứng.
- Giải pháp này làm tăng đáng kể hiệu suất nhưng yêu cầu sử dụng thêm về tại nguyên không gian bộ nhớ.

III – Collision

Kỹ thuật xử lý va chạm – **Các phương pháp kết nối (Chaining)**



Collisions Handled by Forming a Linked-List at each index

III – Collision

Kỹ thuật xử lý va chạm – **tăng kích thước mảng hoặc DSLK**

- Sử dụng ngưỡng, sau khi vượt qua ngưỡng thì tạo CTDL có kích thước gấp đôi kích thước ban đầu. Sau đó thực hiện sao chép phần tử cũ vào CTDL mới
- Phương pháp này làm giảm đáng kể xung đột nhưng cần cẩn thận chọn ngưỡng. Thường là 0.6, có nghĩa là khi 60% kích thước của bảng băm đã được lấy đầy, ta thực hiện việc thay đổi kích thước.

III - Collision

Kỹ thuật xử lý va chạm – **phương pháp băm kép (double hashing)**

- Trong hàm băm kép có hai hàm băn con. Hàm băm thứ hai sử dụng để điều chỉnh kết quả của hàm băm thứ nhất khi gây ra xung đột.
- Hàm băm kép có thể tìm thấy vị trí trống tiếp theo trong bảng băm so với cách dò tuyết tính.

```
(firstHash(key) + i * secondHash(key)) % tableSize
```

Bài tập

Mô tả việc sử dung hash table cho các trường hợp sau

1. Tìm kiếm danh bạ
2. Ngăn chặn việc nhập trùng dữ liệu
3. Sử dụng hash table bộ nhớ đệm khi duyệt website

THANK YOU FOR YOUR ATTENTIONS