Bit Operators

Int → 32 bit

00000000 00000000 00000000 00000000

Int 32 bits this is how its represented. Recall this to base 10?

$$\text{base 10} = \sum_{i=0}^{31} 2^i \cdot v_i$$

ex) 0110

$= 2^3 \cdot 0 + 2^2 \cdot 1 + 2^1 \cdot 1 + 2^0 \cdot 1$

$= 4 + 2 = 6$

$2 \cdot (0110)_2 = 2\left[2^3 \cdot 0 + 2^2 \cdot 1 + 2^1 \cdot 1 + 2^0 \cdot 1\right]$

$= 2^4 \cdot 0 + 2^3 \cdot 1 + 2^2 \cdot 1 + 2^0 \cdot 1$

$= 8 + 4 = 12$

What happens to bits when we multiply by 2? shifts to the left.

$=_2 1100$

C we call left shift operator <<

```
int x = 73;
x = x << 1;      x <<= 1; // multiplies x by 2.
```

Shift all bits in x by 1 to the left.

```
int y = 10;
y <<= 3 //
```

What's the value of y?

$[1010$

$2^3 \left[2^3 \cdot 1 + 2^1 \cdot 1\right]$

$\rightarrow 2^6 \cdot 1 + 2^4 \cdot 1$

$64 \quad 16 = 80$

In general

```
intx;
x <<= y → x · 2^y
```

Signed Numbers and one and two's compliment

Two's compliment of a number is obtained by adding 1 to its compliment.

→ bitwise negation

4 bit

compliment 1010 → 0101 → ~x

2's compliment 1010 → 0101 → 0110

---

ex) 30 = 16 + 8 + 4 + 2 =

0000 0000 0001 1110 : 30 in base 2.

~30 1111 1111 1110 0001

+1

2's c 1111 1111 1110 0010 ← -30

3 A positive number in a system that is 2's compliment is written in base 2. A negitive number is the compliment + 1.

Right Shift (>>)

0110 ← 6

Shift 1

0010 ← 2

1001 ← -6

Shift add 1 because it had 1 a one the → 1101 ← 1100 → 0011 → -3

Bit Masks

Int message; // bits 1-10 the id#

10-13 are a flag for

→ 14-20 → price

20- are meaningless

0 → no error
1 → error
2 → fire error
3 → system
4 → error 3

```
struct S
    int id;
    int → Error error_type;
    float price;
```

sizeof 256 bit>

2^9 max 3 2^2 2^6 3

```
int sendMessage( int pd, int code, int price) {
    int m = 0;          // 0000 0000 0000 0000 0000 0000
    price;              //                    0000 0000 1010
    price <<= 14;       // 0000 0010 1000 0000 0000;
    mt = price;         // 0000 0010 1000 0000 0000;
    code <<= 10;        //                1
    mt = code;          // 0000 0010 1000 1000 0000;        16+8+4+2
    id <<=1;            // 0000 0000 0000 0011 1100;
    mt = id;            // 0000 0010 1000 1011 1100;
    return m;
```

void readMessage(int m, int *id, int *error_code, int *price){

// bits 10-1 I want.

```
0101 1011   &
0001 1100   mask
---------
0001 1000
```

int id_mask = $2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7 + 2^8 + 2^9 + 2^{10}$; (1-10)

*id = (m & id_mask) >> 1

price_mask = $2^{14} + 2^{15} + 2^{16} + 2^{17} + 2^{18} + 2^{19} + 2^{20}$;

*price = (m & price_mask) >> 14;

error_code_mask = $2^{11} + 2^{12} + 2^{13}$

*error_code = (m & error_code_mask) >> 11;

0000 1111 0000

0000 0111 0000

4-6 are the # of items.

(0000 0111 0000) → $2^4 + 2^5 + 2^6$

then shift right 4,

0000 0000 0111 → $2^0 + 2^1 + 2^2$

Struct bitpack

unsigned int pad2 : 1

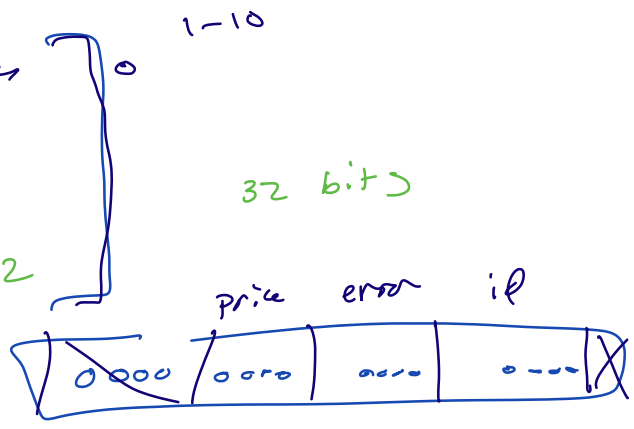
unsigned int id : 9

unsigned int error_code : 3

unsigned int price : 6

unsigned int pad2 : 12

3



struct bitpack readMessage(int m)

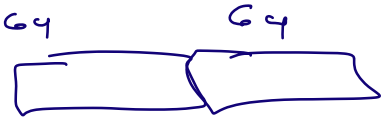
return (bitpack) m;

3

P.id →

```
struct Book {
    char* title;
    char* author
}

struct ByOrder {
    unsigned int price : 3
}
```



```
int stringHash(char* str) {
    int h = 0;
    while (*str != '\0') {
        h += (int)(*str);
        str = str + 1;
    }
    return h % n;
}
```

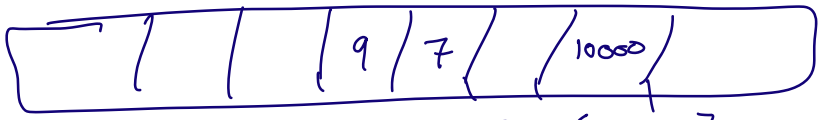
↖ size of hash table.

```
struct Book {
    char* title;
    char* author;
}

int BookHash(Book* b) {
    return strHash(title) + strHash(author);
}
```

Implementation

First Linear Probing



Key | value | Hash

4 . 7 4

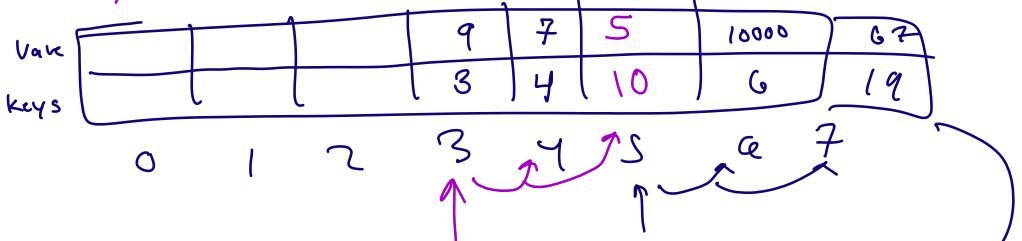
3 → 9 3

10 → 5 3

6 10000 6

100000

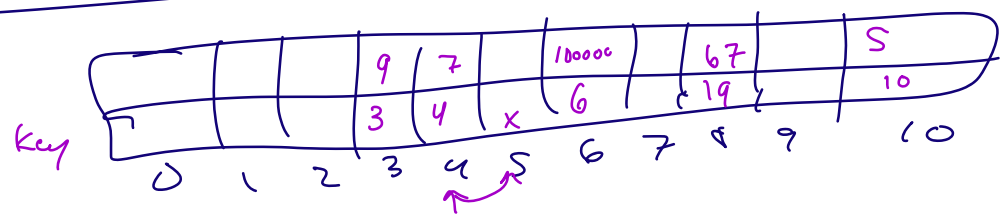
Linear Probing



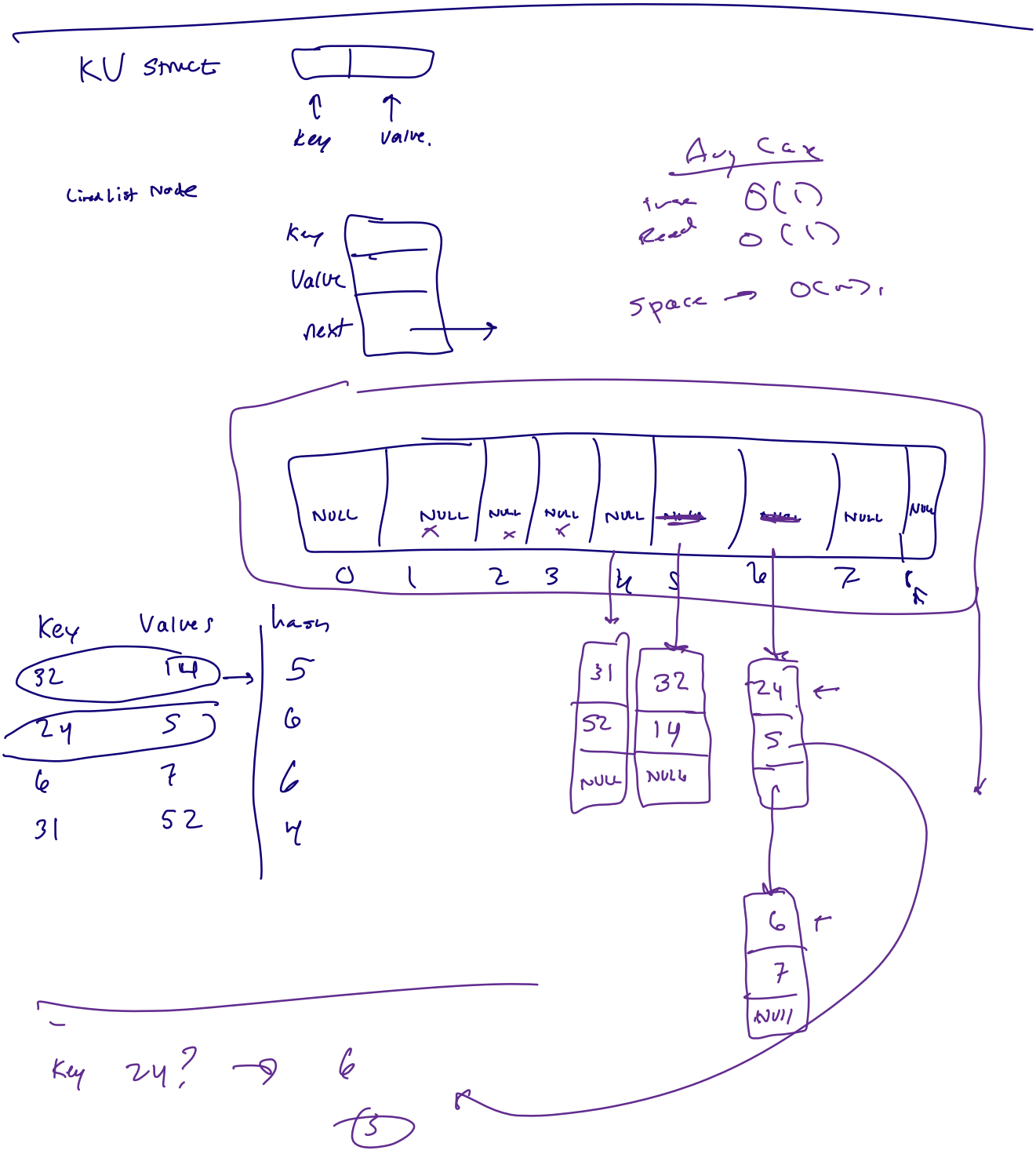
insert (19, 67)

↓

hash → 5



Key → 15 hash → 4 → no value for key 15


KV struct
Key
Value.
Any cax
time O(1)
Read O(1)
space → O(n).
Linked list Node
Key
Value
next
NULL
NULL
NULL
NULL
NULL
NULL
NULL
0 1 2 3 4 5 6 7
Key Values
32 14
24 5
6 7
31 52
hash
5
6
6
4
31
52
NULL
32
14
NULL
24
5
6
7
NULL
Key 24? → 6
5