hashtable

April 16, 2021

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]

[1]:
```
ord('h')
```

[1]: 104

[2]:
```
for i in map(ord, "hello world"):
    print(i)
```

```
104
101
108
108
111
32
119
111
114
```

```
108
100
```

[3]:
```
sum(map(ord,'hello world'))
```

[3]: 1116

[4]:
```
sum(map(ord,'world hello'))
```

[4]: 1116

[5]:
```
sum(map(ord,'gello xorld'))
```

[5]: 1116

Different Key - Same hash Value. so we add a multiplier in our logic, add the values obtained by multpling ordinal value of character and character position.

[6]:
```
def myhash(s):
    mult = 1
    hv = 0
    for ch in s:
        hv += mult * ord(ch)
        mult += 1
    return hv
```

[7]:
```
print(myhash('hello world'))
print(myhash('world hello'))
print(myhash('gello xorld'))
```

```
6736
6616
6742
```

[8]:
```
print(myhash('ad'))
print(myhash('ga'))
```

```
297
297
```

0.0.1 Still collision happened.

Better go for resolving collisions rather than creating perfect hash function.

0.0.2 Class for Hash item

class to hold hash table items.

{key - value }

[9]:
```
class HashItem:
    def __init__(self, key, value):# constructor
        self.key = key
        self.value = value
```

0.0.3 Class for Hash Table

0.0.4 Hash table with linear probing as conflict addressing method.

[10]:
```
class HashTable:
    def __init__(self): # constructor for the class
        self.size = 256 #size of the HT, count of the table
        self.slots = [None for i in range(self.size)]  # list comprehension
        self.count = 0 # initialised with zero

    def _hash(self, key): #method inside the HT class...   _ access specifier,␣
↪protected private and public
        mult = 1
        hv = 0
        for ch in key:
            hv += mult * ord(ch)
            mult += 1
        return hv % self.size # compression code

    def put(self, key, value):# arg - key and value
        item = HashItem(key, value) # hash item object
        h = self._hash(key) # hash value from the key

        while self.slots[h] is not None:  # slot empty?  ...
            if self.slots[h].key is key:  # check for collision
                break
            h = (h + 1) % self.size
        if self.slots[h] is None:
            self.count += 1
        self.slots[h] = item

    def get(self, key): # arg is key ...
        h = self._hash(key) # calc hash value
        while self.slots[h] is not None: # slot empty??
            if self.slots[h].key is key: # check collision
                return self.slots[h].value
            h = (h+ 1) % self.size
        return None

    def __setitem__(self, key, value): # dunder methods , magic methods ._str_
        self.put(key, value)
```

```
    def __getitem__(self, key):
        return self.get(key)
```

[11]:
```
ht = HashTable() # create a HT obj..
```

[12]:
```
ht.put("good","eggs")
ht.put("better","ham")
ht.put("best","spam")
ht.put("ad","do not")
ht.put("ga","collide")
ht.put("data","value")
```

[13]:
```
for key in ("good", "better", "best", "worst", "ad", "ga"):
    v = ht.get(key)
    print(v)
```

```
eggs
ham
spam
None
do not
collide
```

[14]:
```
print("The number of elements is: {}".format(ht.count))
```

```
The number of elements is: 6
```

[15]:
```
ht["name"]= "sarath"
```

[16]:
```
ht["name"]
```

[16]:
```
'sarath'
```

Additional Link