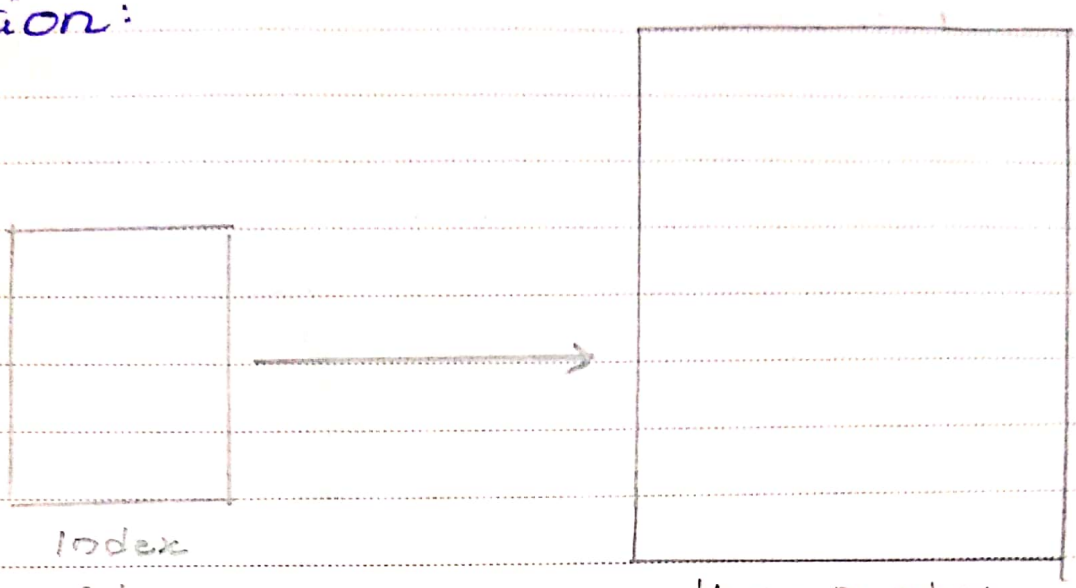
PAGE NO.: PH
DATE: / /

Motivation:



What if index file is so huge that it does not fit in RAM?

↳ Create index for an index and so on.

Can we think of a faster access?

Insert is O(1)

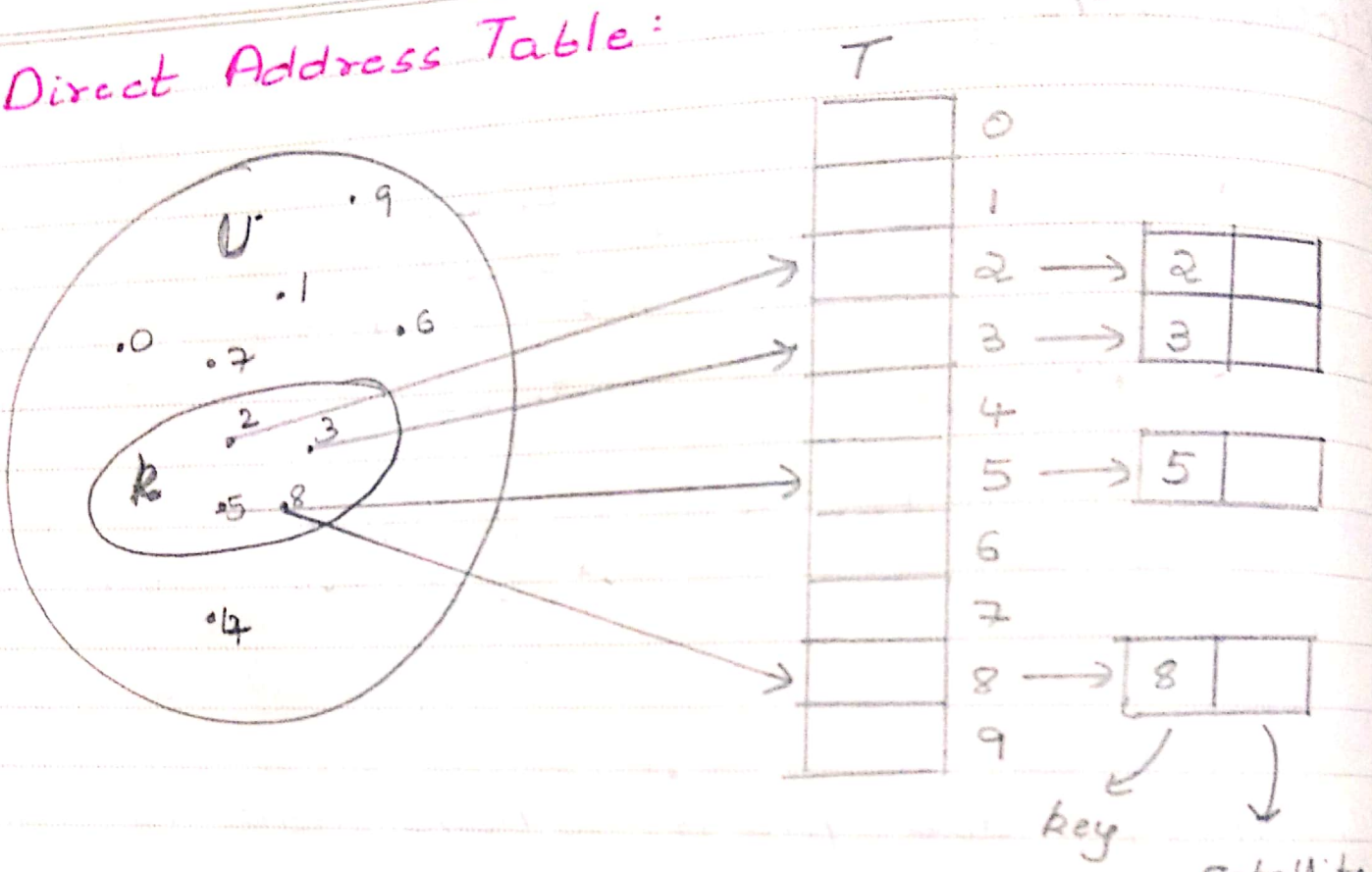
Delete is O(1)

Search is O(1)

↳ Can we achieve this?

DATE : / /

Direct Address Table:

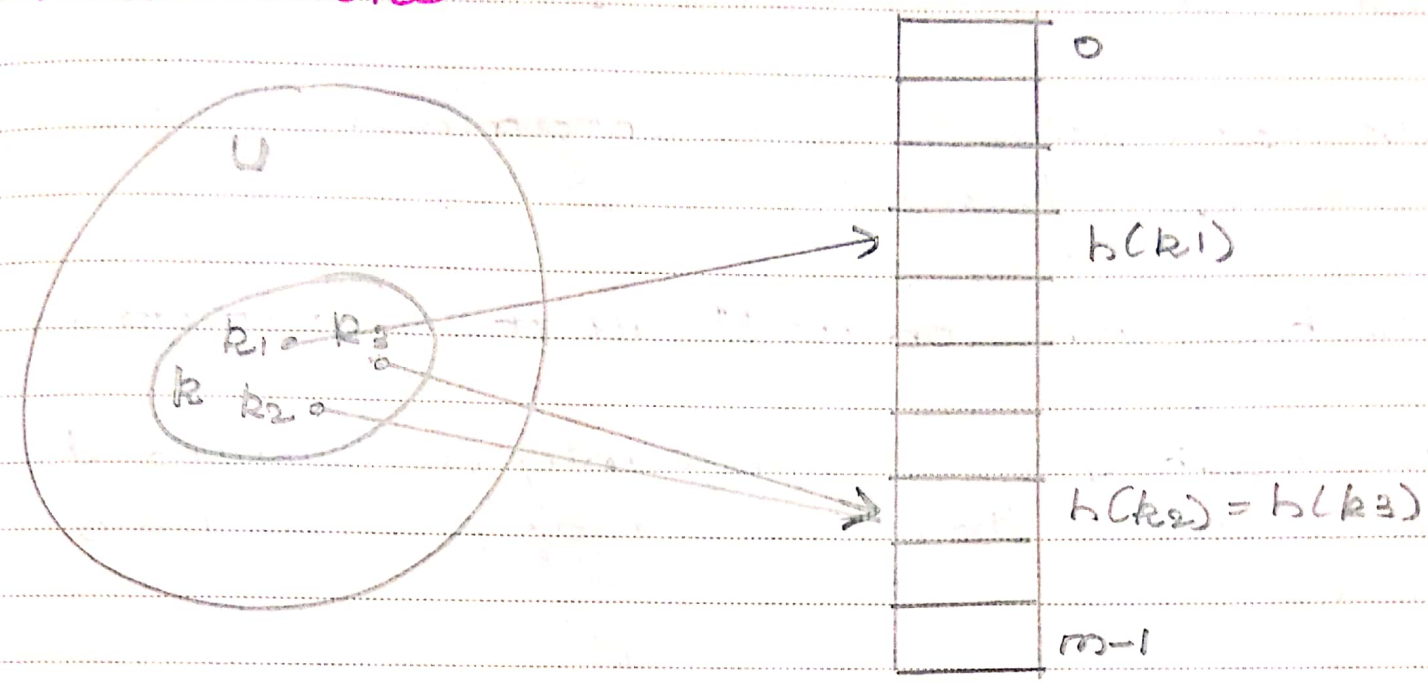


Properties:

- Each slot corresponds to key in Universe
- We can as well store directly the satellite data.
- Like a dictionary
- Universe can have similar items, which are not usually only integers
- We dont have an infinite table size.

PAGE NO. :
DATE : / /

Hash Tables:



Properties:

- Each key goes through a hash function to generate a value for the given key

- $h(k)$
  $h : U \rightarrow \{0, 1 \ldots m-1\}$

- Reduces the array indices that need to be handled.

- Two keys may hash to same slot
  - Collision

- The tradeoff between
  Space & time

- Designing $h(k)$ is a challenge.

Hash Function Properties:

1. The hash value is fully determined by the data being hashed.
2. The hash function uses all the input data
3. The hash function "uniformly" distributes the data across the entire set of possible hash values
4. The hash function generates very different hash values for similar strings.

All of the above, if not adhered to, leads to collision.

Hash function example:

1.
```
int hash( char *str, int m)
{
    int sum;
    if (str == NULL)
        return -1;
    for ( ; *str; str++ )
        Sum = Sum + *str;

    return sum % m;
}
```

1. Satisfies
2. Satisfies
3. Breaks
4. Breaks

2. Jenkins hash:

```
uint32_t jenkins_one_at_a_time_hash (const
    uint8_t * key, size_t length)
{   ↳ unsigned 8 bit integer
    size_t i = 0;
    uint32_t hash = 0;
    while (i != length)
    {
        hash += key[i++];
        hash += hash << 10;
        hash ^= hash >> 6;
    }
    hash += hash << 3;
    hash ^= hash >> 11;
    hash += hash << 15;

    return hash;
}
```

size_t
↳ return type of sizeof()

one_at_a_time ("a", 1)
0xca2e9442.
↳ Sample value

PAGE NO. :
DATE : / /

3. lose lose (terrible hashing algorithm)

```
unsigned long hash (unsigned char *str)
{
    unsigned int hash = 0;
    int c;

    while (c = *str++)
        hash = hash + c;
    return hash;
}
```

4. djb2

```
unsigned long hash (unsigned char *str)
{
    unsigned long hash = 5381;
    int c;

    while (c = *str++)
        hash = ((hash << 5) + hash) + c;

    return hash;
}
```

hash * 33 + c

Magic number 33, one of the best hash functions for strings.

Why 33 works has never been explained, adequately.

PAGE NO. :
DATE : / /

5. unsigned hash (char *s)
{
    unsigned hashval;
    for (hashval = 0; *s != '\0', s++)
        hashval = *s + 31 * hashval;

    return hashval;
}

6. In a hash table of size 13 which index positions would the following two keys 27 & 130 map to?

a) 1, 10 b) 13, 0 c) 1, 0
Answer.
d) 2, 3.

27 % 13 = 1 & 130 % 13 = 0

Exercise:

Design a hash function for:
Input : SRN
output : hash index
Table size : 1000

- SRN is a mixture of integers & chars.
- all data from SRN needs to be used.