PICT, PUNE

Roll. No. 21118
Date: 21-May-21

Subject: Data structure & algorithms
Assignment No. 04
Hashing Techniques-I

Problem Definition:
Consider telephone book database of N clients. Make use of hash table implementation to quickly look up client's telephone no. Make use of two collision handling techniques & compare them using number of ~~collisions~~ comparisons required to find a set of telephone numbers (use linear poking with ~~replacement~~ & without replacement.)

Objectives:
- To understand about hashing & various hashing techniques.
- To study various collision handling techniques (linear poking) & compare them.

Outcomes:
- Implement linear poking as a collision handling technique in hashing.
- Use python oop features.

Hardware requirements
Manufacturer: Acer
System type: x-64 type PC
Processor & RAM: Intel(R) core i5-8265U @ 1.60 GHz & 8 GB of RAM (512 GB ROM)

Software Requirements:
Operating system: MS windows-10 (home edition)
IDE: eclipse (2020 edition)

PICT, PUNE

Theory:

1) Hashing is process of converting a given key into another value.
A hash function is used to generate the new value according to mathematical algorithm.

2) Collisions: Two keys can generate same hash. This is called collision

3) Applications: Hashing is used to store data in hash tables, hash table has wide applications in field of computer science.

4) Collision handling techniques :
   a) Closed Addressing
   b) Open Addressing.
      i) Linear Probing
      ii) Quadratic Probing
      iii) Double hashing

Time complexities:

Inserting: Average case: O(1)
worst case: O(n)

Deletion: Average case: O(1)
worst case: O(n)

Searching: Average case: O(1)
worst case: O(n)

ADT:

```
class Record():
    def __init__(self, num, name):    // constructor.
        self.ph.num = num,
        self.name = name.
```

PICT, PUNE

```
    --str--(self):  // while printing the object.
        print(ph.num + name)

class HashTable():
    def getHashVal(); // hashing function, returns
                        hash value of key.
    def search(key); // search key in hashtable.
                     // return no. of comparisons if key
                        found else return -1
    def insert(key,value): // insert key in hash table.
    def delete(key,value): // delete key from hash table
```

Pseudocodes:

```
search(key): // return no. of comparisons if key found
             // else return -1
    if size of hashtable is zero:
        return -1
    idx ← getHashVal(key)
    init ← idx, comp ← 0.
    while(1):
        comp ← comp+1
        if table[idx] == key:   // key found in table.
            return comp.
        if table[idx] == -1 and this slot is
            not marked as deleted:
            return -1
        idx ← idx+1
        if(idx == init:   // traversed complete table.
            return -1
```

PICT, PUNE

```
insert(key,val): // on successful inserting return True
                    else return false.
  If(hashtable is full (or) key is already
     present in hashtable)
     return false.

  idx ← getHashValue(key)
  while table[idx] != -1: // look through table
      idx ← idx+1           until empty slot is not
  table[idx] = Record(key,val) found
  size ← size+1
  return 1
```

insert with replacement:

```
insert(key, val):
  If(hashtable is full (or) key is already
     present in hashtable)
     return false.

  idx ← getHashVal(key)
  if table[idx] is empty
      table[idx] ← Record(key,val)
  elif idx == getHashVal(table[idx]):
      while table[idx] is not empty:
          idx ← idx+1
      table[idx] = Record(key,val)
      size ← size+1
  else
      tmp = table[idx]
      table[idx] = Record(key,val) // Key replacement
```

```
while table[idx] is not empty:
    idx ← (idx+1)
table[idx] = tmp
```

delete(key): // return True on successfull deletion, false on unsuccessfull.

```
if key is not present in hashtable
    return 0.
idx = getHashVal(key):
while True:
    if table[idx] == key:
        break
    idx ← (idx+1)
table[idx] = -1      // delete key
is_del[idx] = True   // mark slot as deleted
return 1
```

Hashfunction:

hash function = key % N.

where N is the size of the hash table.

PICT, PUNE

Testcases:

1) without replacement.

Operation	Expected output	Actual Output	Verdict.
insert 22,33,42,44 with their values as 'A','B','C','D'	keys should be inserted at 2,3,4,5 positions	keys are inserted at respective positions	Passed.
Search 44	44 should be found with 2 comparisons.	44 is founded & 2 comparisons were required	Passed
Delete key 42 from hash table	42 should be deleted	42 is deleted. (on searching 42 NOT FOUND message is printed)	Passed.

2) With replacement.

Same keys are inserted as of without replacement case.

On searching key 44, it is found with 1 comparison whereas it was found with 2 comparisons in without replacement case.

Analysis of Algorithms:

	Operation	Time complexity	Space complexity
1)	Insertion	Average case: O(1) Worst case: O(n)	O(1)
2)	Searching	Average case: O(1)	O(1)

PICT, PUNE

3) Deletion	Worst case: O(n) Average case: O(1) Worst case. O(n)	O(1)

Applications:

Hashing has many real-world applications including computer architecture & cryptography.

Some of them are mentioned below:

1) Hashtables data structure
2) Compiler Designing
3) Some algorithms like Rabin-Karp work on basis of hashing concept
4) Password verification

etc.

Conclusion:

In this assignment we have learned about various hashing techniques & implemented Linear probing (with & without replacement) as a collision handling technique.