Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.
Department of AI-ML

Experiment	9
Aim	Implement the given problem statement
Objective	Hash functions - Implement hash functions with different collision resolution techniques.
Name	Balla Mahadev Shrikrishna
UCID	2023300010
Class	A
Batch	A
Date of Submission	24-10-24

Explanation of the technique used	Linear Probing :

Q. Array size = 10

Collision count = ?

Density of list after all insertions = ?

Insert using modulo, collision resolution with open addressing, linear probing.

144467, 234534, 214562, 224562, 137456, 214576, 140145, 199645, 162145

Soln:

h(144467) = 144467 % 10 = 7

h(234534) = 234534 % 10 = 4

h(214562) = 214562 % 10 = 2

h(224562) = 224562 % 10 = 2 ⇒ 1st collision (1)

1st probe = (2+1) % 10 = 3

h(137456) = 137456 % 10 = 6

h(214576) = 214576 % 10 = 6

⇒ 2nd collision (2)

1st probe = (6+1) % 10 = 7 → 1

2nd probe = (6+2) % 10 = 8

162145	0
	1
214562	2
224562	3
234534	4
140145	5
137456	6
144467	7
214576	8
199645	9

h(140145) = 140145 % 10 = 5

h(199645) = 199645 % 10 = 5

⇒ 3rd collision (3)

1st probe = (5+1) % 10 = 6 → (1)

2nd probe = (5+2) % 10 = 7 → (2)

3rd probe = (5+3) % 10 = 8 → (3)

4th probe = (5+4) % 10 = 9

h(162145) = 162145 % 10 = 5

⇒ 4th collision (4)

1st probe = (5+1) % 10 = 6 → (1)

2nd probe = (5+2) % 10 = 7 (2)

3rd probe = (5+3) % 10 = 8 (3)

4th probe = (5+4) % 10 = 9 (4)

5th probe = (5+5) % 10 = 0

Density ⇒ Load Factor = total no. of occupied cells / total size of table

No. of collisions = 4 .... {excluding probes}

Density = $\frac{9}{10}$ = **0.9**

No. of collisions including probes = 12

Quadratic Probing :

Q. quadratic probing.
elements same as that of the prev. ques.

Soln: $h(144467) = 144467 \% 10 = 7$
$h(234534) = 234534 \% 10 = 4$
$h(214562) = 214562 \% 10 = 2$
$h(224562) = 224562 \% 10 = 2 \Rightarrow 1^{st}$ collision
1^{st} probe $= [2+(1)^2] \% 10 = 3$
$h(137456) = 137456 \% 10 = 6$
$h(214576) = 214576 \% 10 = 6 \Rightarrow 2^{nd}$ collision
1^{st} probe $= [6+(1)^2] \% 10 = 7 \longrightarrow 1$
2^{nd} probe $= [6+(2)^2] \% 10 = 10 \% 10 = 0$
$h(140145) = 140145 \% 10 = 5$
$h(199645) = 199645 \% 10 = 5 \Rightarrow 3^{rd}$ collision
1^{st} probe $= (5+1^2) \% 10 = 6 \longrightarrow 1$
2^{nd} probe $= (5+2^2) \% 10 = 9$
$h(162145) = 162145 \% 10 = 5 \Rightarrow 4^{th}$ collision
1^{st} probe $= (5+1^2) \% 10 = 6 \longrightarrow 1$
2^{nd} probe $= (5+2^2) \% 10 = 9 \longrightarrow 2$
3^{rd} probe $= (5+3^2) \% 10 = 4 \longrightarrow 3$
4^{th} probe $= (5+4^2) \% 10 = 1$

214576	0
162145	1
214562	2
224562	3
234534	4
140145	5
137456	6
144467	7
	8
199645	9

No. of collisions excluding probes = 4
No. of collisions including probes = 9

Double Hashing :

Q. $h(k) = k \% 19$
$g(k) = 7 - (k \% 7)$

Soln: $h(144467) = 144467 \% 19 = 10$
$h(234534) = 234534 \% 19 = 17$
$h(214562) = 214562 \% 19 = 14$
$h(224562) = 224562 \% 19 = 1$
$h(137456) = 137456 \% 19 = 10 \Rightarrow 1^{st}$ collision
1^{st} probe $= [10 + (7-4)] \% 19$
$= 13$
$h(214576) = 214576 \% 19 = 9$
$h(140145) = 140145 \% 19 = 1 \Rightarrow 2^{nd}$ collision
1^{st} probe $= [1 + (7-5)] \% 19$
$= 3 \% 19$
$= 3$
$h(199645) = 199645 \% 19$
$= 5$
$h(162145) = 162145 \% 7 = 4$

	0
224562	1
	2
140145	3
162145	4
199645	5
	6
	7
	8
214576	9
144467	10
	11
	12
137456	13
214562	14
	15
	16
234534	17
	18

Program(Code)

```
#include<stdio.h>
#include<stdlib.h>

// #define hashSize 10 //for linear and quadratic
#define hashSize 19 //for double hashing
```

```
typedef struct hashNode{
    int index;
    char status;
    int val;
}hashNode;

hashNode *hashTable[hashSize] = {NULL};

hashNode *createNode(int index, char status, int val){
    hashNode *new = (hashNode *)malloc(sizeof(hashNode));
    new->index = index;
    new->status = status;
    new->val = val;
    return new;
}

int h(int k){
    return k % hashSize;
}

int linearProbing(int val){
    int idx = h(val);
    while(hashTable[idx] != NULL && hashTable[idx]->status != 'E'){
        if(hashTable[idx]->status == 'D'){
            return idx;
        }
        idx = (idx + 1) % hashSize;
    }
    return idx;
}

int quadraticProbing(int val){
    int idx = h(val);
    int i=1;
    while(hashTable[idx] != NULL && hashTable[idx]->status != 'E'){
        if(hashTable[idx]->status == 'D'){
            return idx;
        }
        idx = (h(val) + (i*i)) % hashSize;
        i++;
    }
    return idx;
}

int h2(int k) {
    return k % hashSize;
}

int g(int k) {
```

	<pre> return 7 - (k % 7);
	}
	
	int doubleHashing(int val){
	    int idx = h2(val);
	    int i=1;
	    while(hashTable[idx] != NULL && hashTable[idx]->status != 'E'){
	        if(hashTable[idx]->status == 'D'){
	            return idx;
	        }
	        idx = (h(val) + (i*g(val))) % hashSize;
	        i++;
	    }
	    return idx;
	}
	
	void insert(int val, int x){
	    int idx;
	    switch(x){
	        case 1:
	            idx = linearProbing(val);
	            break;
	
	        case 2:
	            idx = quadraticProbing(val);
	            break;
	
	        case 3:
	            idx = doubleHashing(val);
	            break;
	
	        default:
	            printf("Invalid..");
	            return;
	    }
	
	    if(hashTable[idx] == NULL || hashTable[idx]->status == 'E'){
	        hashTable[idx] = createNode(idx, 'O', val);
	    }
	    else if(hashTable[idx]->status == 'D'){
	        hashTable[idx]->val = val;
	        hashTable[idx]->status = 'O';
	    }
	}
	
	void printHash(hashNode **hashTable){
	    for(int i=0; i<hashSize; i++){
	        if(hashTable[i] == NULL || hashTable[i]->status == 'E'){
	            printf("Loc %d : -1\n", i);
	        }</pre>

	<pre> else{
	            printf("Loc %d : %d\n", i, hashTable[i]->val);
	        }
	    }
	}
	
	void freeHash(){
	    for(int i=0; i<hashSize; i++){
	        if(hashTable[i]!=NULL){
	            free(hashTable[i]);
	            hashTable[i] = NULL;
	        }
	    }
	}
	
	int main(){
	    int n=0, temp=0, x=0;
	
	    printf("Enter the number of elements : ");
	    scanf("%d", &n);
	
	    printf("Choose the probing method -\n1. Linear Probing\n2.
	Quadratic Probing\n3. Double Hashing\nEnter your choice : ");
	    scanf("%d", &x);
	
	    printf("Enter the elements -\n");
	    for(int i=0; i<n; i++){
	        scanf("%d", &temp);
	        insert(temp, x);
	    }
	
	    printHash(hashTable);
	    freeHash();
	    return 0;
	}</pre>

Output	Linear Probing :

hash.c

hash.c > insert1(int)

```c
int main(){
    printf("Enter the number of elements : ");
    scanf("%d", &n);
    printf("Enter the elements -\n");
    for(int i=0; i<n; i++){
        scanf("%d", &temp);
        insert1(temp);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
students@spit:~/2023300010$ gcc hash.c
students@spit:~/2023300010$ ./a.out
Enter the number of elements : 9
Enter the elements -
144467
234534
214562
224562
137456
214576
140145
199645
162145
Loc 0 : 162145
Loc 1 : -1
Loc 2 : 214562
Loc 3 : 224562
Loc 4 : 234534
Loc 5 : 140145
Loc 6 : 137456
Loc 7 : 144467
Loc 8 : 214576
Loc 9 : 199645
students@spit:~/2023300010$
```

Quadratic Probing :

```
hash.c
hash.c > ...
int quadraticProbing(int val){
    while(hashTable[idx] != NULL && hashTable[idx]->status != 'E'){
        if(hashTable[idx]->status == 'D'){
            return idx;
        }
        idx = (h(val) + (i*i)) % hashSize;
        i++;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Mahadev\S.E\DS\Lab Sessions> gcc hash.c
PS C:\Mahadev\S.E\DS\Lab Sessions> ./a.exe
Enter the number of elements : 9
Choose the probing method -
1. Linear Probing
2. Quadratic Probing
3. Double Hashing
Enter your choice : 2
Enter the elements -
144467
234534
214562
224562
137456
214576
140145
199645
162145
Loc 0 : 214576
Loc 1 : 162145
Loc 2 : 214562
Loc 3 : 224562
Loc 4 : 234534
Loc 5 : 140145
Loc 6 : 137456
Loc 7 : 144467
Loc 8 : -1
Loc 9 : 199645
PS C:\Mahadev\S.E\DS\Lab Sessions>
```

Double Hashing :

```
hash.c ×
hash.c > main()
121    int main(){
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Mahadev\S.E\DS\Lab Sessions> ./a.exe
Enter the number of elements : 8
Choose the probing method -
1. Linear Probing
2. Quadratic Probing
3. Double Hashing
Enter your choice : 3
Enter the elements -
123456
234567
345678
456789
567890
678901
789012
890123
Loc 0 : -1
Loc 1 : -1
Loc 2 : -1
Loc 3 : -1
Loc 4 : -1
Loc 5 : -1
Loc 6 : 789012
Loc 7 : -1
Loc 8 : -1
Loc 9 : -1
Loc 10 : 456789
Loc 11 : 345678
Loc 12 : 234567
Loc 13 : 123456
Loc 14 : 678901
Loc 15 : 890123
Loc 16 : -1
Loc 17 : -1
Loc 18 : 567890
PS C:\Mahadev\S.E\DS\Lab Sessions>
```

Conclusion

In this experiment, we successfully implemented hash functions with three collision resolution techniques: Linear Probing, Quadratic Probing, and Double Hashing. Linear Probing demonstrated straightforward implementation, while Quadratic Probing offered reduced clustering. Double Hashing, using a secondary hash function, proved to be the most efficient in distributing values across the table. Overall, these techniques highlighted the importance of proper collision resolution in optimizing hash table performance.