← CLOSED HASHING →

date 03/6/2023

* Closed Hashing → Also called Linear Hashing
  - Recall - uses a fixed space for storage & thus limits the size of the sets (an array data structure)

Ex: Inserting elements

- Set A = {a, b, c, d, e, f}

• Insert	• Hash Values
a	H(a) = 3 ✓
b	H(b) = 9 ✓
c	H(c) = 4
d	H(d) = 3 ✓
e	H(e) = 9 ✓
f	H(f) = 0

Dictionary D

0	e
1	f
2	
3	a
4	c
5	d
6	
7	
8	
9	b

* Insert() is O(1) since it goes directly to the location.

TERMINOLOGY

- Synonyms - elements w/ the same hash value
  - ex: a & d are synonyms
  - results in COLLISION
- Collision - when an element is inserted in an already occupied space.
  - Solution: linear Hashing
- Displacement - when an element is inserted in an already occupied space by a non-synonymous member
  - ex: f & e.
  - solution: Linear Hashing.
- Linear Hashing - a solution to collision wherein the element is placed in the next available position in circular array.
  - uses % operator

Formula: $H_i(x) = (H(x) + i) \% MAX$

- Example: H(d) = 3 → i = 1 $H_1(d) = (H(d) + 1) \% 10 = (3+1) \% 10 = 4$ ✗ occupied
  i = 2 $H_2(d) = (H(d) + 2) \% 10 = (3+2) \% 10 = 5$ ✓ not occupied

← CLOSED HASHING →

date _____

• Operations:

1.) initialize() - set each cell of dictionary to EMPTY.

2.) member() - search for element stops when:

> element is found

> EMPTY cell is encountered

> number of components = MAX

↳ use counter variable

3.) insert() - elements must be UNIQUE but two different elements can have the same hash value.

- inserts elements at EMPTY / DELETED cell but priorities SEARCH LENGTH.

4. delete() - when element is found, mark as DELETED

• Advantages		• Disadvantages	
Closed Hash	Open Hash	Closed Hash	OH
- exact location	- Open Space	- Collisions occur	- Unable to O(1)
- O(1)	- no collisions		

★ EMPTY & DELETED

↳ When calling the isMember() function, for closed hashing it is O(1) since the hash() func. returns the exact location

↳ The function:

> STOP search at empty slot

> CONTINUE search when it encounters a deleted elem.

Hence, differentiate EMPTY & DELETED using macros:

```
#define EMPTY 0
#define DELETED -1
```

• NOTE: these markers should be the SAME datatypes as the elements in the dictionary.

Strathmore

← CLOSED HASHING →

date

• Exercise: Average Search Length

• Formulas: Search Length

SL = Actual location of x - Hash(x) + 1 of x

Average Search Length

Ave SL = Sum of SL / no. of elements / MAX

- SL is if dictionary is circular / doesn't rotate
- Ave SL is to see if hash func. is efficient & correct.

Hash Values	Dictionary D	
	0	EMPTY
	1	EMPTY
H(A) = 1	2	EMPTY
H(B) = 4	3	EMPTY
H(C) = 9	4	EMPTY
H(D) = 9	5	EMPTY
H(E) = 0	6	EMPTY
H(F) = 3	7	EMPTY
H(G) = 4	8	EMPTY
H(H) = 3	9	EMPTY

Do the ff.

1.) Insert the elements A, B, C, D, E, F, G, and H in an initially empty dictionary w/ hash values 1, 4, 9, 9, 0, 3, 4, and 3 respectively. Note: solution for collision is linear Hashing

2.) Determine the search length of each elem.

3.) Determine the Average Search length

• Solution

	D
0	E
1	A
2	F
3	H
4	B
5	G
6	EMPTY
7	EMPTY
8	EMPTY
9	C

• SL

A = (1-1)+1 = 1

B = (4-4)+1 = 1

C = (9-9)+1 = 1

D = (0-9)+1 = -8 (actually 2)

E = (2-0)+1 = 3

F = (3-3)+1 = 1

G = (5-4)+1 = 2

H = (6-3)+1 = 4

15

• Ave SL

= 15/8 = 1.875

Strandmore

<- CLOSED HASHING ->

date _____

- The "perfect" hash func. ()
  - → returns a unique value for each element
  - → has no collisions, no synonyms, is O(1)

★ Performance Evaluation of Hashing ★

- Load factor / packing density – ratio of no. of elements to be stored to no. of available spaces
  - rule of thumb: 80%
    - ↳ more space = less likely for collisions/synonyms

Note: Packing density & collision are: inversely proportional (!)

- Packing density formula: # of elems / x = 80%
  - or
  - x / 0.80
  - ~~where x is the no. of spaces~~

- Solutions to Collisions -

1.) Linear Hashing - most common

2.) Group synonyms in one location - Var 3
   - ↳ can be: LL, CBL

3.) Double Hashing

← CLOSED HASHING →

Prac ex. 4 date 03/06/2023

• Illustration:

Hash vals.		Dictionary D
	0	e
H(a)=3	1	f
H(b)=9	2	EMPTY
H(c)=4	3	a
H(d)=3	4	DELETED
H(e)=9	5	d
H(f)=0	6	EMPTY
H(g)=1	7	EMPTY
H(j)=5	8	EMPTY
	9	

1.) define EMPTY and DELETED

2.) Write an appropriate definition of datatype Dictionary

(1.)
```
#define EMPTY ' '
#define DELETED '!'
#define MAX 10
```

(2.)
```
typedef char Dictionary[SIZE];
```

3.) Write the codes of the ff. operations: a.) initialize, b.) member, c) insert, d.) delete

a.) initialize

• Illustration:

	Dictionary D
0	EMPTY
1	EMPTY
2	EMPTY
3	EMPTY
4	EMPTY
5	EMPTY
6	EMPTY
7	EMPTY
8	
9	

• Code:

```
void initDict(Dictionary D){
    int index;
    for(index=0; X<MAX; X++){
        D[X] = EMPTY;
    }
```

b.) Member

```
int isMember(Dictionary D, char elem
    int hashVal = hash(elem);
    int temp, retval;
    if (D[hashVa] != x){
for (temp=hashVal +1; temp != hashVal && D[temp] != elem; temp =(temp+1)%MAX){}
    }
return retval=(temp != hashval) ? 0 : 1;
}
```

Study more

← Closed Hashing

date _____

c) Insert()

- Code:

```
void insertDict(Dictionary D, char elem){
    int hashVal = hash(elem);
    int temp;
    if(D[hashVal] != elem){
    for (temp = hashVal; isMember(D, D[temp]) != 1 &&
                D[temp] != EMPTY && D[temp] != DELETED;
                temp = (temp + 1) % MAX){}
        D[temp] = elem;
    }
}
```

d) delete()

- Code:

```
Void deleteDict(Dictionary D, char elem){
    int hashVal = hash(elem);
    int temp = hashVal;
    if (D[temp] != elem){
        for (temp = hashVal + 1; temp != hashVal && D[temp] != X;
            temp=(temp + 1) % MAX){}
    }
    D[temp]= DELETED;
}
```

← CLOSED HASHING VARIATIONS →

date _____

★ **Variation 3** - the most efficient / "semi-open-hashing"

- Linking synonyms except that the synonym area cells are linked together during initialization. Last variable is changed to AVAIL

// initialization:

Note:

synonym area implementation is similar to cursor-based

Dictionary D

0	EMPTY		10		11
1	EMPTY		11		12
2	EMPTY		12		13
3	EMPTY		13		14
4	EMPTY		14		15
5	EMPTY		15		16
6	EMPTY		16		17
7	EMPTY		17		18
8	EMPTY		18		19
9	EMPTY		19		-1
/* prime data area */			/* Synonym Area */		

10

Avail

// Simulation:

Note: non-synonym elems have link: -1

Dictionary D

0	f	-1	10	d	-1
1	EMPTY		11	e	-1
2	EMPTY		12	g	10
3	g	12	13		14
4	c	-1	14		15
5	EMPTY		15		16
6	EMPTY		16		17
7	EMPTY		17		18
8	EMPTY		18		19
9	b	11	19		-1
/* prime data area */			/* synonym area */		

13

Avail

- If element has a synonym, it will be inserted in the synonym area, with link node updated to -1 to indicate its the last link to original element in prime data area, which will have its link node updated to link to synonym.
- Will end up looking like an open hashing imp.
- For deleting(), dont change link of element because it needs to be connected, just mark as DELETED & return the slot to Avail.

<- Closed Hashing Var ->

date _____

- Code:

```
#define MAX 20
#define EMPTY '!'
#define DELETED '!'

typedef struct {
    char data;
    int link;
} nodeType;

typedef struct {
    nodeType Nodes[MAX];
    int Avail;
} Dictionary;

void initialize(Dictionary *D) {
    int index; D->Avail = MAX/2;
    for (index=0; index<D->Avail; index++); // init Prime data area
        D->Nodes[index].data = EMPTY;
    }
    for (index=D->Avail; index<MAX; index++) {
        D->Nodes[index].link = index+1;
    }
    D->Nodes[MAX-1].link = -1;
}
```

date _____

// insert()

```
void insert (Dictionary* D, char elem){
int hashVal, temp;
hashVal = hash(elem);
IF (D→Nodes[hashVal].data == EMPTY ||
    D→Nodes[hashVal].data == DELETED){
    D→Nodes[hashVal].data = elem;
    D→Nodes[hashVal].link = -1;
} else{
    //insertFirst
    temp = D→Nodes[D→Avail].link;
    D→Nodes[D→Avail].data = elem;
    D→Nodes[D→Avail].link = D→Nodes[hashVal].link;
    D→Nodes[hashVal].link = D→Avail;
    D→Avail = temp;
}
}
```

date _____

NdeleteE()

```
void deleteElem(Dictionary *D, char elem){
  int hashVal = hash(elem);
  int temp;
  if (D->Nodes[hashVal].data != EMPTY){
    if (D->Nodes[hashVal].data == elem){
      D->Nodes[hashVal].data = DELETED;
    } else {
      for (temp = D->Nodes[hashVal].link;
        temp != -1 && D->Nodes[temp].data != elem;
        temp = D->Nodes[temp].link){}
      if (temp != -1){
        D->Nodes[temp].data = DELETED;
        D->Nodes[temp].link = D->Avail;
        D->Avail = temp;
      }
    }
  }
}
```