UNIT-V

Hashing:GeneralIdea,HashFunctions, CollisionResolution-SeparateChaining, OpenAddressing- Linear probing, Quadratic Probing, Double Hashing, Rehashing, ExtendibleHashing,ImplementationofDictionaries.

****

Hashing:

- Hashing is one of the searching techniques that uses a constant time. The time complexity in hashing is **O (1).**
- Tillnow, we read the two techniques for searching, i.e., linear search and binary search. The worst time complexity **in linear search is O(n), and O(logn) in binary search**. In all these searchtechniques,asthenumberofelementsincreasesthetimerequiredto searchanelement also increases linearly.
- Usinghashingdatastructure,agivenelementissearchedwith **constanttime complexity**.
- Hashing is an effective way to reduce the number of comparisons to search an element in a data structure.

<u>Hashingis definedasfollows...</u>

"Hashingistheprocessofindexingandretrievingelement(data)inadatastructureto provide a faster way of finding the element using a hash key"

- Here,thehashkeyisavaluewhichprovidestheindexvaluewheretheactualdataislikelyto bestoredinthedatastructure. Inthisdatastructure,weuseaconceptcalled**Hash table**.
- **Hash table:Hash table** to store data. All the data values are inserted into the hash tablebased on the hash key value.
- The hashkeyvalue is usedto mapthe datawithan index orslot inthehashtable. Andthe hash key is generated for every data using a **hash function.**
- That means every entry in the hash table is based on the hash key value generated using the hash function.
- Hash tables are used to perform insertion, deletion and search operations very quickly in adata structure. Using hash table concept, insertion, deletion, and search operations are accomplished in constant time complexity. Generally, every hash table makes use of a function called **hash function** to map the data into the hash table.

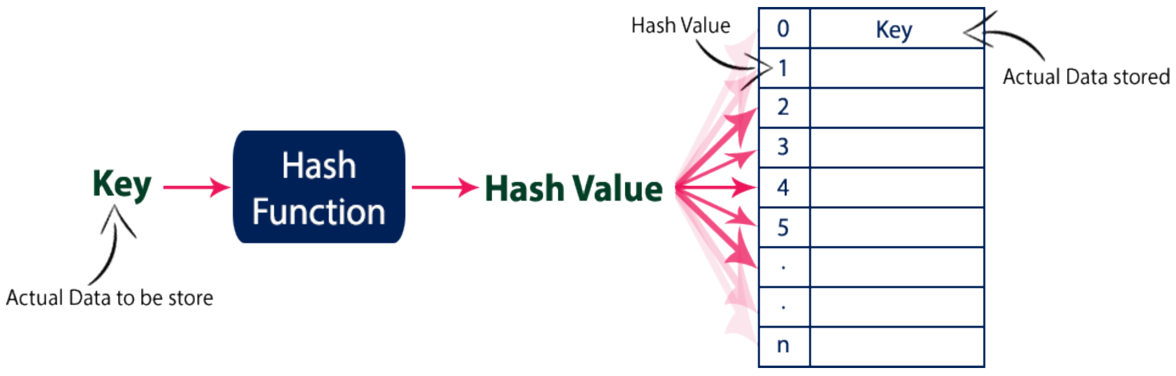
<u>HashTableisdefinedas follows...</u>

"Hashtableisjustanarraywhichmapsakey(data)intothedatastructurewiththe help of hash function such that insertion, deletion and search operations are performedwithconstanttimecomplexity(i.e.O(1))."

❖ <u>A**hashfunction**isdefinedasfollows...</u>

Hashfunctionisafunctionwhichtakesapieceofdata(i.e.key)asinputandproduces an integer (i.e. hash value) as output which maps the data to a particular index in the hash table.

Basicconceptofhashingandhashtableisshowninthefollowingfigure...



Example:

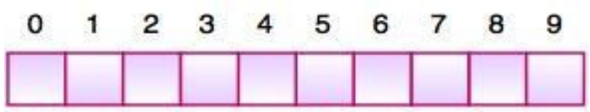


Fig. Hash Table

➢ The above figure shows the hashtablewiththesize ofn =10.Eachposition ofthehashtable is called as**Slot or Index**. In the above hash table, there are n slots in the table, names = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. Slot 0, slot 1, slot 2 and so on. Hash table contains no items, so every slot is empty.

➢ As we know the mapping between an item andthe slot where item belongs in the hash table is called the hash function. The hash function takes any item in the collection and returns an integer in the range of slot names between 0 to n-1.

Example:01

❖ Suppose we have integer items {26, 70, 18, 31, 54, and 93}. One common method of determining a hash key is the division method of hashing and the formula is :

❖ Division methodorremindermethodtakes an item anddivides it bythe table size andreturns the remainder as its hash value.

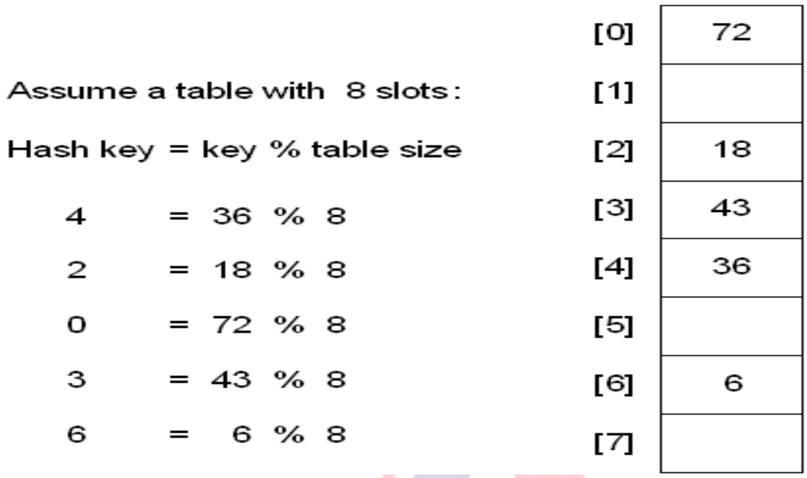
DataItem	Value%No.ofSlots	HashValue
26	26% 10=6	6
70	70% 10=0	0
18	18% 10=8	8
31	31% 10=1	1

54	54% 10=4	4
93	93% 10=3	3
41	**41% 10=1**	**1(collision occurs)**
74	**74% 10= 4**	**4(collision occurs)**

Fig. Hash Table

Example:02

Supposewehaveintegeritems{36,18,72,43, and6}.

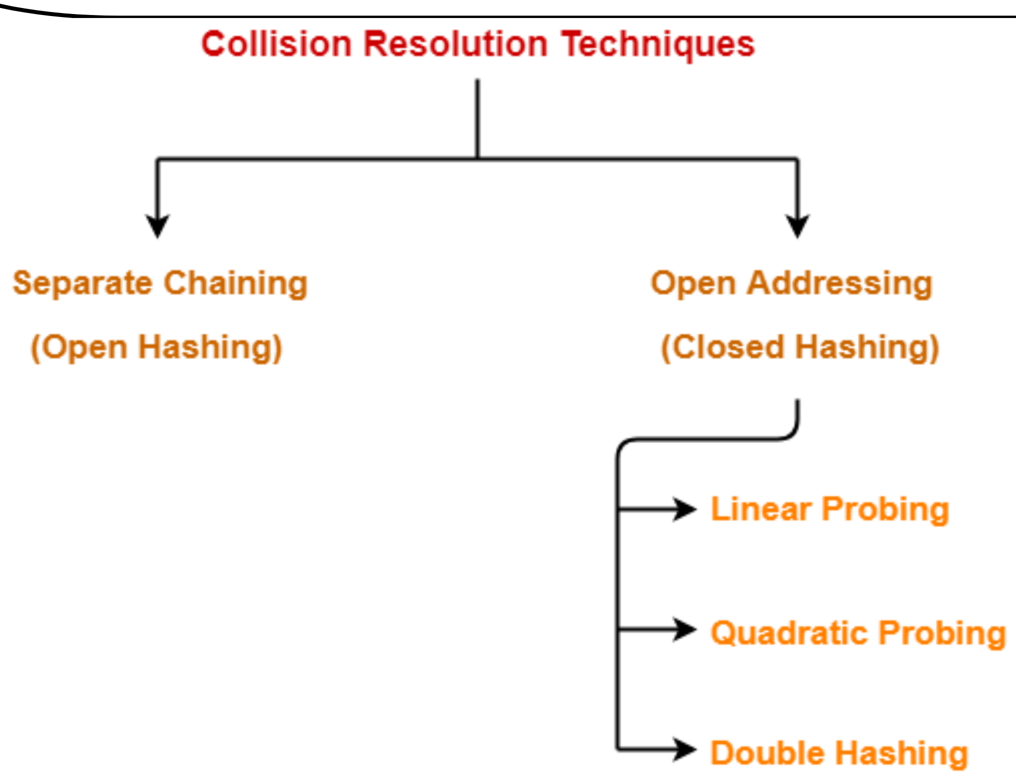


CollisionResolutionTechniques

CollisioninHashing-

1. Inthis,thehashfunctionisusedtocomputetheindexofthearray.
2. Thehashvalue isusedtostorethe keyinthehashtable,asanindex.
3. Thehashfunctioncanreturnthesame hashvaluefortwoormorekeys.
4. When two or more keys are given the same hash value, **it is called a collision.** To handlethis collision, we use collision resolution techniques.

Whenthehashvalueofakeymapstoanalreadyoccupiedbucket/slot/indexofthe hash table,it is called as a Collision.



Separatechaining(openhashing):

Tohandlethe collision,

- Thistechniquecreatesalinkedlisttotheslotfor whichcollision occurs.
- Thenewkeyistheninsertedinthelinkedlist.
- Theselinkedliststotheslotsappearlikechains.
- Thatiswhy;thistechniqueiscalledas **separatechaining**.

Timecomplexity:

- Itsworst-casecomplexityforsearchingis o(n).
- Itsworst-casecomplexityfordeletioniso(n).

Advantagesofseparatechaining

- Itiseasyto implement.
- Thehashtableneverfillsfull,sowecanaddmoreelementstothechain.
- Itisless sensitiveto thefunctionofthe hashing.

Disadvantagesofseparatechaining"

- Inthis,cacheperformanceofchainingisnotgood.
- Thememory wastageistoomuchinthismethod.
- Itrequiresmorespaceforelementlinks.

LoadFactor(λ)

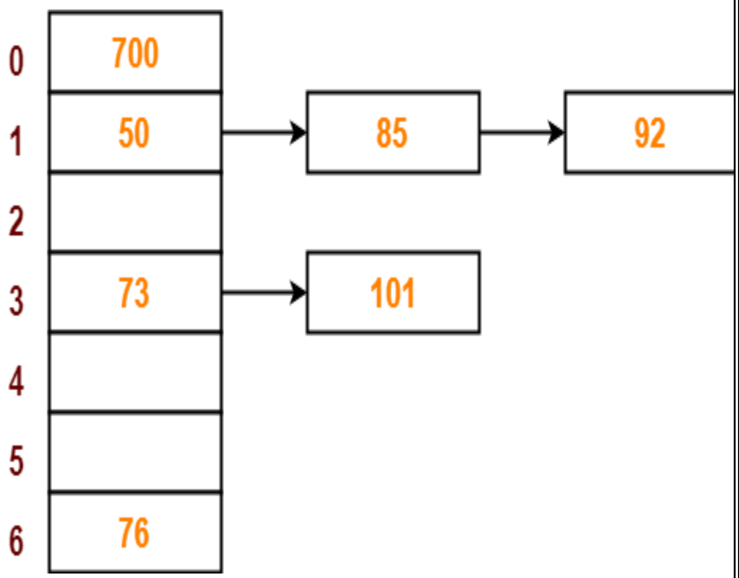
Loadfactor(λ)isdefinedas-

Loadfactor(λ)= NumberofElementspresentinthehashtable(n) / TotalsizeoftheHashTable(N)

Example:01

Usingthehashfunction'keymod7',insertthefollowingsequenceofkeysinthehashtable- 50, 700, 76, 85, 92, 73 and 101.Use separate chaining technique for collision resolution.

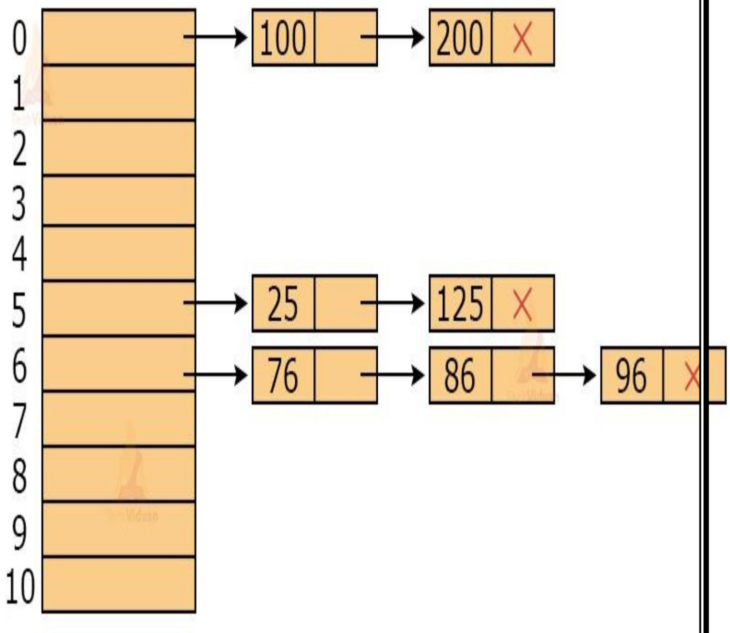
DataItem	Value%No.of Slots	HashValue
50	50% 7=1	1
700	700%7 =0	0
76	76% 7=6	6
85	85% 7=1	1 (collisionoccurs)
92	92% 7=1	1 (collisionoccurs)
73	73% 7=3	3
101	101%7=3	**3** (collisionoccurs)



Example:02

Let the keys be 100, 200, 25, 125, 76, 86, 96 and let m = 10. Given, h(k) = k mod 10 .Use separate chaining technique for collision resolution.

DataItem	Value%No.of Slots	HashValue
100	100 % 10=0	0
200	200 % 10=0	0 **(collisionoccurs)**
25	25 %10 =5	5
125	125 % 10=5	5 **(collisionoccurs)**
76	76 %10 =6	6
86	86 %10 =6	6 **(collisionoccurs)**
96	96% 10=6	**6 (collisionoccurs)**



Openaddressing(closedhashing)

Open addressing is collision-resolution method that is used to control the collision in the hashing table. There is no key stored outside of the hash table. Therefore, the size of the hash table is always greater than or equal to the number of keys. It is also called **closed hashing**.

1.Linearprobing:

- This Hashing technique finds the hash key value through hash function and maps the keyon particular position in hash table.

  key=key%size;

- In case ifkey has same hashaddress (collision) then it will find **next emptyposition** in the hash table.

  key= (key+i)%size;hereiis0tosize-1

- We take the hash table as circular array. So if table size is N then after N-1 position it willsearch from 0^{th} position in the array.

Example:01:

Supposewehavealistofsize20(m=20).Wewanttoputsomeelementsinlinearprobing fashion. The elements / keys are {96, 48, 63, 29, 87, 77, 48, 65, 69, 94, 61}

DataItem	Value%No.ofSlots	HashValue	probes
96	96%20=16	16	1
48	48%20=8	8	1
63	63%20=3	3	1
29	29%20=9	9	1
87	87%20=7	7	1
77	77%20=17	17	1
48	48%20=**8** (48+1)%20=**9** (48+2)%20=10	**Collisionoccurs** **10**	**3**
65	65%20=5	5	1
69	69%20=**9** (69+1)%20=**10** (69+2)%20=**11**	**Collisionoccurs** **11**	**3**
94	94%20=14	14	1
61	61%20=1	1	1

HashTable

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	61		63		65		87	48	29	48	69			94		96	77		

- TheorderofElements/keysinHashTableare:
  --,61,--,63,--,65,--87,48,29,48,69,--,--,94,--,96,77,--,--
- TotalNumberofProbesare **15**

Example:02:

Supposewehavealistofsize11(size=11).Wewanttoputsomeelementsinlinearprobing fashion.Theelements/keyare{29,18,43,10,36, 25,46}

DataItem	Value%No.ofSlots	HashValue	probes
29	29% 11=7	7	1
18	18% 11=**7** (18+1)%11=8	**Collisionoccurs** **8**	2
43	43% 11=10	10	1
10	10%11 =**10** (10+1)%11=0	**Collisionoccurs** **0**	2
36	36% 11=3	3	1
25	25% 11=**3** (25+1)%11=4	**Collisionoccurs** 4	2
46	46% 11=2	2	1

HashTable

0	**10**
1	
2	**46**
3	**36**
4	**25**
5	
6	
7	**29**
8	**18**
9	
10	**43**

- TheorderofElementsinHashTableare:10,--,46,36,25,--,--,29,18,--,43
- TotalNumberofProbesare **10**

Example:03 (Assignment)

Supposewehavealistofsize10(size=10).Wewanttoputsomeelements/keysinlinear probing fashion. The elements /Keys are {18,41,22,32,44,59 }.

Advantage-

- Itiseasyto compute.

Disadvantage-

- Themainproblemwithlinearprobingis**clustering**.
- Manyconsecutiveelementsform groups.
- Then,ittakestimetosearchanelementortofindanempty bucket.

2.Quadratic Probing:

- This Hashing technique finds the hash key value through hash function and maps the keyon particular position in hash table.

key=key%size;

- In case ifkey has same hashaddress (collision) then it will find **next emptyposition** in the hash table.

key=(key+i²)%size; hereiis0tosize-1

- We take the hash table as circular array. So if table size is N then after N-1 position it willsearch from 0^{th} position in the array.

Example:01

Suppose we have a list of size 20 (size = 20). We want to put some elements / keys inQuadraticProbing fashion. The elements /Keys are { 96, 48, 63, 29, 87, 77, 48, 65, 69, 94, 61}.

DataItem	Value%No.ofSlots	HashValue	probes
96	96% 20=16	16	1
48	48% 20=8	8	1
63	63% 20=3	3	1
29	29% 20=9	9	1
87	87% 20=7	7	1
77	77% 20=17	17	1
48	48% 20=**8** (48+1)%20=**9** (48+4)%20=12	**Collisionoccurs** **12**	**3**
65	65% 20=5	5	1
69	69% 20=**9** (69+1)%20=**10**	**Collisionoccurs** **10**	**2**

94	94% 20=14	14	1
61	61% 20=1	1	1

HashTable

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	61		63		65		87	48	29	69		48		94		96	77		

- TheorderofElements/keysinHashTableare:--,61,--,63,--,65,--87,48,29,69,--,48,--94,--96,77,--,--
- TotalNumberofProbesare **14**

Example:02

Suppose we have a listof size11(size = 11).We want to put someelements/ keys inQuadratic Probing fashion. The elements /Keys are {29, 18, 43, 10, 46, and 54}

DataItem	Value%No.ofSlots	HashValue	probes
29	29% 11=7	7	1
18	18% 11=**7** (18+1)%11=8	**Collisionoccurs** **8**	2
43	43% 11=10	10	1
10	10%11 =**10** (10+1)%11=0	**Collisionoccurs** **0**	2
46	46% 11=2	2	1
54	54%11=**10** (54+1)%11=**0** (54+4)%11=**3**	**Collisionoccurs** **3**	3

HashTable

0	**10**
1	
2	**46**
3	**54**
4	
5	
6	
7	**29**
8	**18**
9	
10	**43**

- TheorderofElementsinHashTableare:10,--,46,54,--,--,--,29,18,--,43
- TotalNumberofProbesare **10**

Example:03 (Assignment):

Supposewehavealistofsize10(size=10).We wanttoputsomeelements/keysinQuadraticProbing fashion. The elements /Keys are {18,41,22,32,44,59

3.DoubleHashing:

- ❖ Doublehashingusestheideaofapplyingasecondhashfunctiontokeywhencollision occurs.
- ❖ Firsthashfunctiontypically**h1 (key)=key%size**
- ❖ Doublehashingcanbedoneusing(WhencollisionOccurs):

  (h1(key)+i*h2 (key))% Tablesize.

  Here**h2(key)=PRIME-(key%PRIME)**wherePRIMEisaprimesmallerthanTable Size.
- ❖ Werepeatbyincreasingiwhencollisionoccurs(ivaluesfrom0tosize-1)

NOTE:WhencollisionoccursthenfindstheSecondHashfunction

Example:01

Supposewehavealistofsize10(size=10).Wewanttoputsomeelements/keysinDouble hashing fashion. The elements /Keys are { 3,22,27,40,42,11,19}.

Data Item	h1=h1(key)%10	h2=7-(key%7)	FinalHashvalue =((h1+i*h2))%10	probes
3	3% 10=**3**	-		1
22	22% 10=2			1
27	27% 10=7			1
40	40% 10=0			1
42	42% 10=**2** **CollisionOccurred**	7-(42%7)=7	(2+1*7)%10=**9**	2
11	11% 10=1			1
19	19%10=**9** **Collision Occurred**	7-(19%7)=2	(9+1*2)%10=**1** **Collision Occurred** (9+2*2)%10=**3** **CollisionOccurred** (9+3*2)%10=**5**	4

HashTable

0	**40**
1	**44**
2	**22**
3	**3**
4	
5	**19**
6	
7	**27**
8	
9	**42**

- ➢ TheorderofElementsinHashTableare:40,44,22,3,--,19,--,27,--,42,--.
- ➢ TotalNumberofProbesare **10**

Example:02

Supposewehavealistofsize20(m=20).WewanttoputsomeelementsinDoublehashing fashion. The elements are {96, 48, 63, 29, 87, 77, 48, 65, 69, 94, 61}.

SeparateChainingVsOpenAddressing-

SeparateChaining	Open Addressing
Keysarestoredinsidethehashtableaswell as outside the hash table.	Allthekeysarestoredonlyinsidethe hash table. Nokeyispresentoutsidethehashtable.
Thenumberofkeystobestoredinthehash table can even exceed the size of the hash table.	Thenumberofkeystobestoredinthe hashtablecanneverexceedthesizeof the hash table.
Deletionis easier.	Deletionis difficult.
Extraspaceisrequiredforthepointersto store the keys outside the hash table.	Noextraspaceisrequired.
Cacheperformanceispoor. Thisisbecauseoflinkedlistswhichstorethe keys outside the hash table.	Cacheperformanceisbetter. Thisisbecauseherenolinkedlistsare used.
Somebucketsofthehashtablearenever used which leads to wastage of space.	Bucketsmaybeusedevenifnokeymaps to those particular buckets.

ComparisonofOpenAddressingTechniques-

	LinearProbing	QuadraticProbing	DoubleHashing
Primary Clustering	Yes	No	No
Secondary Clustering	Yes	Yes	No
NumberofProbe Sequence (m=size of table)	m	m	m^2
Cacheperformance	Best	Liesbetweenthe two	Poor

Conclusions-

- LinearProbinghasthebestcacheperformancebutsuffersfrom clustering.
- Quadraticprobingliesbetweenthetwointermsofcacheperformanceand clustering.
- Doublecachinghaspoorcacheperformancebutnoclustering.

Rehashing

Rehashingisatechniqueinwhichthetableisresized,i.e.,thesizeoftableis**doubled**by creatinga new table. It is preferable is the total size of table is a next prime number. There are situations in which the rehashing is required.

- Whentableiscompletelyfull
- WithLinearprobingwhenthetableisfilledwith70%
- Wheninsertionsfailduetooverflow.

❖ Rehashing means**hashing again**. Basically, when the Load Factor Increase or its reaches to 1 or Greater than 1 them complexity Increases.

Loadfactor(λ)= NumberofElementspresentinthehashtable(n) / TotalsizeoftheHashTable (N)

i.e**λ<1**,When**λ=1orλ>1**weneedtoincreasetheHashTableSizethisisKnownas**Rehashing.**

❖ In such situations, we have to transfer entries from old table to the new table by re computing their positions using hash functions.

Example:1

Consider we have to insert the elements 37, 90, 55, 22, 17, 49, and 87. The table size is 10 and will use hash function:

DataItem	Value%No.ofSlots	HashValue	probes
37	37%10=7	7	1
90	90%10=0	0	1
55	55%10=5	5	1
22	22%10=2	2	1
17	17%10=**7** (17+1)%10=8	**Collisionoccurs** 8	2
49	49%10=9	9	1
87	87% 10=**7** (87+1)%10=**8** (87+2)%10=**9** (87+3)%10=**0** (87+4)%10=**1**	**Collisionoccurs** **1**	5

HashTable

0	**90**
1	**87**
2	**22**
3	
4	
5	**55**
6	
7	**37**
8	**17**
9	**49**

❖ Now this table is almost full and if we try to insert more elements collisions will occur and eventually further insertions will fail.

❖ Hence we will rehash by doubling the table size. The old table size is 10 then we should double this size for new table that becomes 20. But 20 is not a prime number, we will prefer to make the table size as 23.

HashTable

Andnewhashfunction willbe

DataItem	Value%No.ofSlots	HashValue	probes
37	37%23=14	14	1
90	90%23=21	21	1
55	55%23=9	9	1
22	22%23=22	22	1
17	17%23=17	17	1
49	49%23=3	3	1
87	87% 23=18	18	1

HashTable

0	
1	
2	
3	**49**
4	
5	
6	
7	
8	
9	**55**
10	
11	
12	
13	
14	**37**
15	
16	
17	**17**
18	**87**
19	
20	
21	**90**
22	**22**

Example:2
Considerwehavetoinserttheelements 7,18,43,10,36,25.Thetablesizeis11.

Extendiblehashing(DynamicHashing)

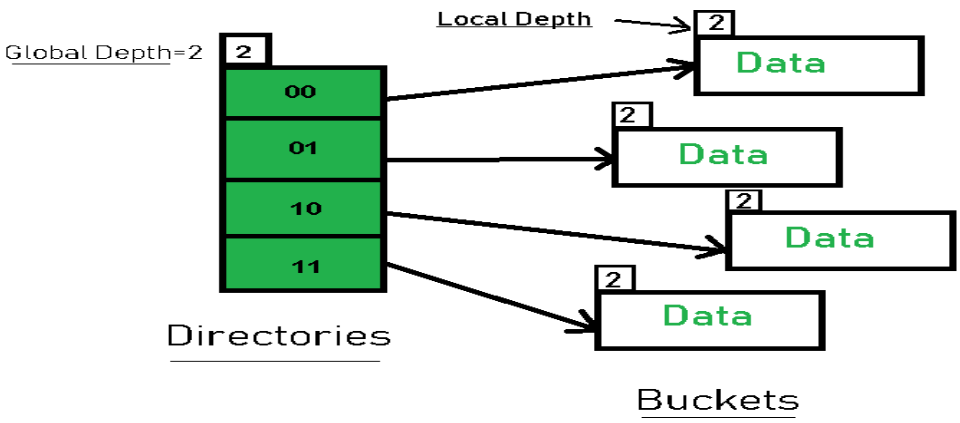
- Inopenaddressingorseparatechaining,themajorproblemiscollisioncouldcauseseveral buckets to be examined to find an alternative empty cell.
- Whenthetablegetstoofilled,arehashingmust beperformedwhichisexpensive.

Extendible hashing:

- Thisisverysimplestrategywhichprovidesquickaccesstimeforinsertsandsearchoperations on large database.
- Itisadynamichashingmethod.The **directories**and**buckets**areusedtohashdata.
- Thecostofexpandingandupdatingisverylow.

- **Directories**: The directories store addresses of the buckets in pointers. An id is assigned toeach directory which may change each time when Directory Expansion takes place.
- **Buckets**:Thebucketsareusedtohashtheactualdata.

BasicStructureofExtendible Hashing:



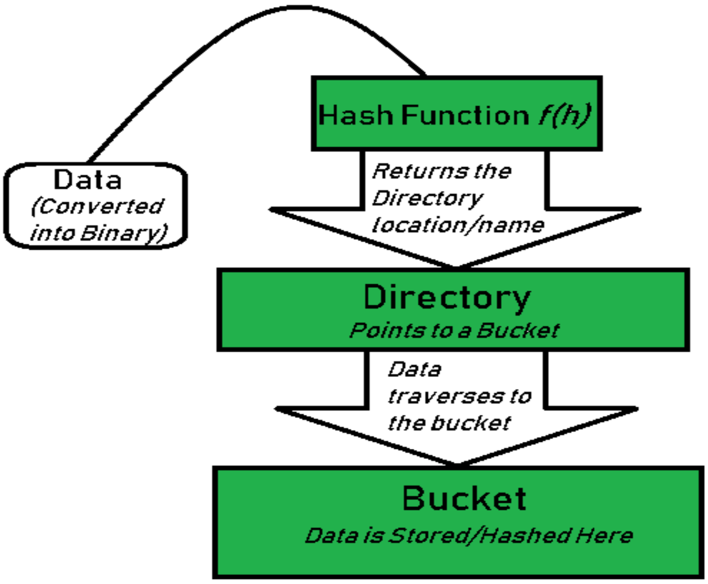
Extendible Hashing

FrequentlyusedtermsinExtendibleHashing:

- **Directories:** These containers store pointers to buckets. Each directory is given a unique id which may change each time when expansion takes place. The hash function returns this directory id which is used to navigate to the appropriate bucket. Number of Directories = 2^Global Depth.
- **Buckets:** Theystorethehashedkeys.Directoriespointtobuckets.Abucketmaycontain more than one pointer to it if its local depth is less than the global depth.
- **Global Depth:** It is associated with the Directories. They denote the number of bits which are usedbythehashfunctiontocategorizethekeys.GlobalDepth=Numberofbitsindirectory id.

- **Local Depth:** It is the same as that of Global Depth except for the fact that Local Depth is associated with the buckets and not the directories. Local depth in accordance with the global depthisusedtodecidetheactionthatto beperformedincaseanoverflowoccurs.Local Depth is always less than or equal to the Global Depth.
- **BucketSplitting:** Whenthenumberofelementsinabucketexceedsaparticularsize,then the bucket is split into two parts.
- **Directory Expansion:** Directory Expansion Takes place when a bucket overflows. Directory Expansion is performed when the local depth of the overflowing bucket is equal to the global depth.

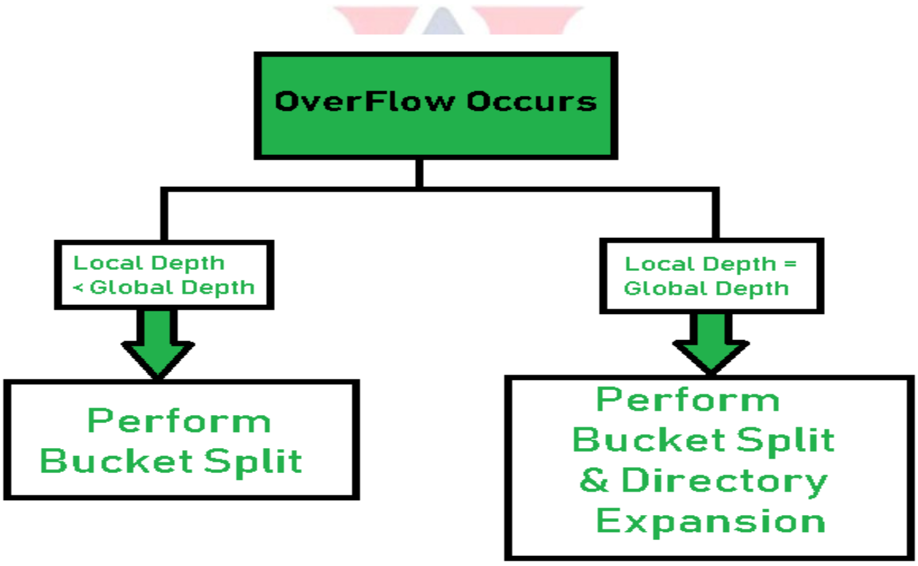
<u>BasicWorkingofExtendibleHashing</u>:



- **Step1–Analyze DataElements:** Dataelementsmay existinvariousformseg.Integer, String, Float, etc.. Currently, let us consider data elements of type integer. eg: 49.
- **Step 2 – Convert into binary format:** Convert the data element in Binary form. For string elements,considertheASCIIequivalentintegerofthestartingcharacterandthenconvert theintegerintobinaryform.Sincewehave49asourdataelement,itsbinaryformis 110001.
- **Step 3 – Check Global Depth of the directory.** Suppose the global depth of the Hash-directory is 3.
- **Step 4 – Identify the Directory:** Consider the 'Global-Depth' number of LSBs in the binary number and match it to the directory id.

  Eg. The binary obtained is: 110001 and the global-depth is 3. So, the hash function will return 3 LSBs of 110**001** viz. 001.
- **Step 5 – Navigation:** Now, navigate to the bucket pointed by the directory with directory-id 001.

- **Step 6 – Insertion and Overflow Check:** Insert the element and check if the bucketoverflows.Ifanoverflowisencountered,goto**step7** followedby **Step8**,otherwise,go to **step 9**.
- **Step 7 – Tackling Over Flow Condition during Data Insertion:** Many times, while inserting data in the buckets, it might happen that the Bucket overflows. In such cases, we need to follow an appropriate procedure to avoid mishandling of data.
  First, Check if the local depth is less than or equal to the global depth. Then choose one of the cases below.
  - **Case1:** If the local depth of the overflowing Bucket is equal to the global depth, then Directory Expansion, as well as Bucket Split, needs to be performed. Then incrementthe global depth and the local depth value by 1. And, assign appropriate pointers. Directory expansions will double the number of directories present in the hash structure.
  - **Case2:** Incasethelocaldepthislessthantheglobaldepth,thenonlyBucketSplit takes place. Then increment only the local depth value by 1. And, assign appropriate pointers.



- **Step8–RehashingofSplitBucketElements:** TheElementspresentintheoverflowing bucket that is split are rehashed w.r.t the new global depth of the directory.
- **Step9–**Theelementissuccessfullyhashed.

Example-1: Now, let us consider a prominent example of hashing the following elements: **16,4,6,22,24,10,31,7,9,20,26.**

BucketSize:3(Assume)

HashFunction: SupposetheglobaldepthisX.ThentheHashFunctionreturnsXLSBs.

Solution:

First, calculate the binary forms of each of the given numbers.

16- 10000

4-00100

6-00110

22-10110

24-11000

10-01010

31-11111

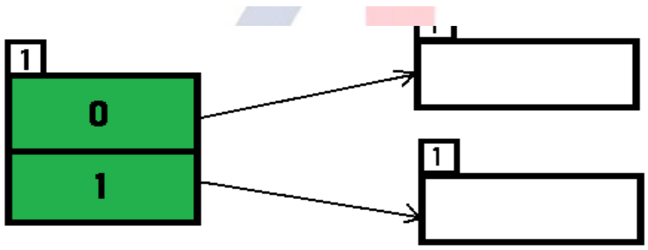
7-00111

9-01001

20-10100

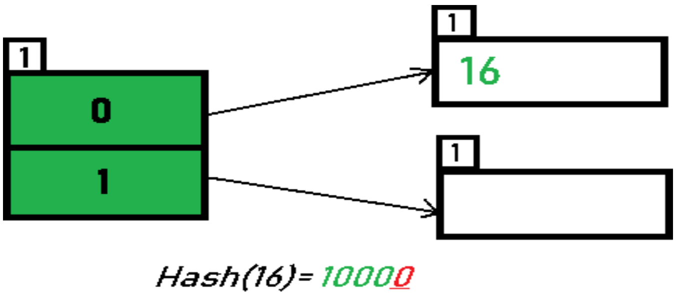
26-11010

➢ Initially, the global-depth and local-depth is always 1. Thus, the hashing frame looks like this:



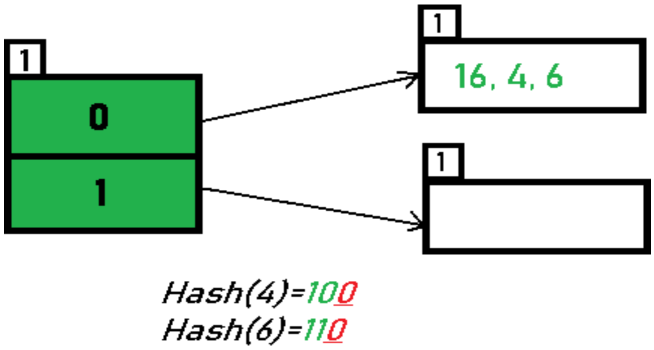
Inserting16:

The binary format of 16 is 10000 and global-depth is 1. The hash function returns 1 LSB of 1000**0** which is 0. Hence, 16 is mapped to the directory with id=0.

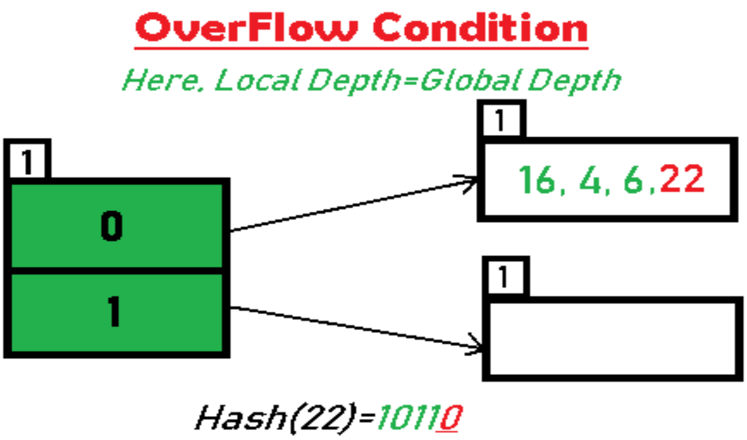


Inserting4and6:

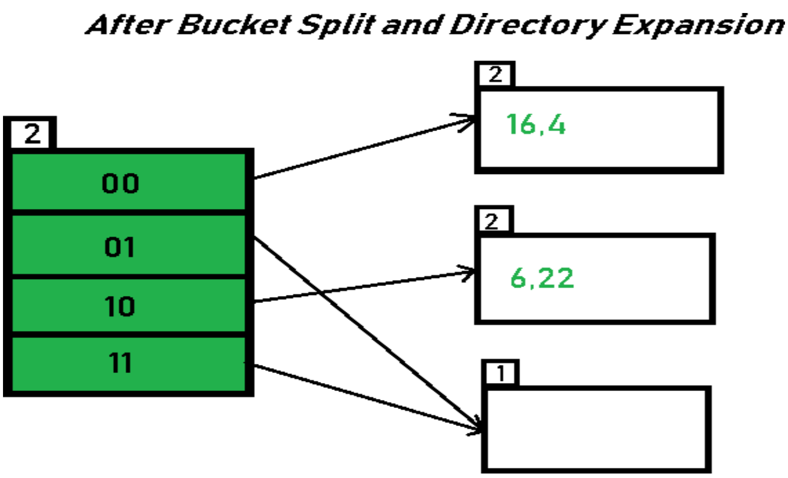
Both4(10**0**)and6(11**0**)have0intheirLSB.Hence,theyarehashedasfollows:



Inserting22: Thebinaryformof22is1011**0**.ItsLSBis0.Thebucketpointedbydirectory 0is already full. Hence, Over Flow occurs.

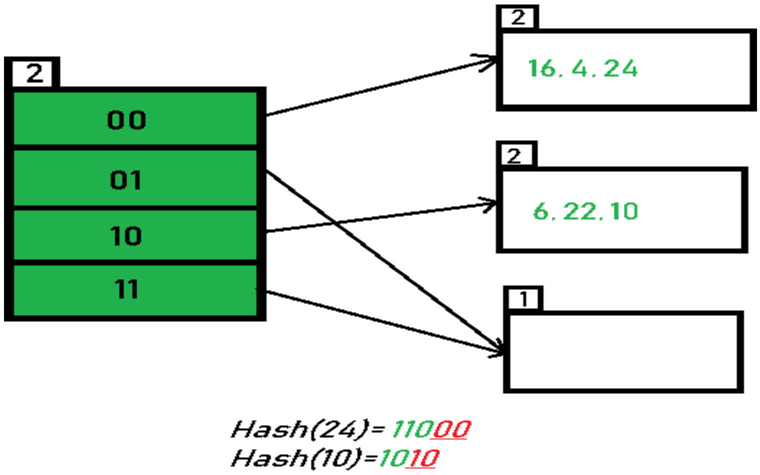


As directed by **Step 7-Case 1**, Since Local Depth = Global Depth, the bucket splits and directory expansion takes place. Also, rehashing of numbers present in the overflowing bucket takes place after thesplit.And,sincetheglobaldepthisincremented by 1,now,theglobaldepthis2.Hence, 16,4,6,22arenowrehashedw.r.t2LSBs.[16(100**00**),4(1**00**),6(1**10**),22(101**10**)].

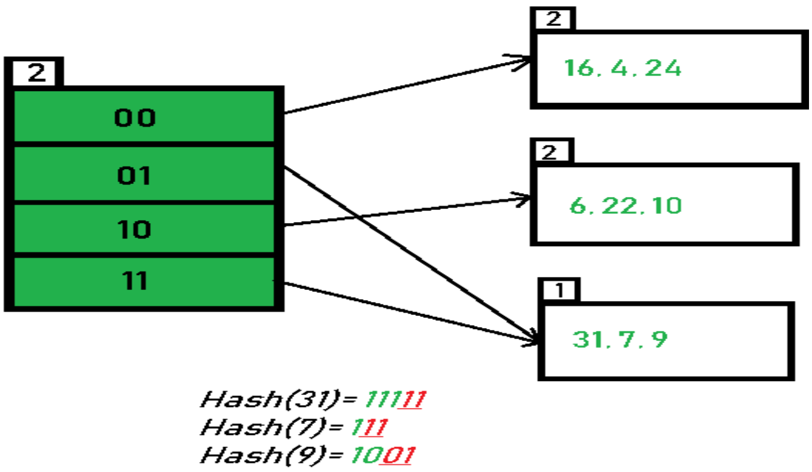


**Notice that the bucket which was underflow has remained untouched. But, since the number ofdirectories has doubled, we now have 2 directories 01 and 11 pointing to the same bucket. This isbecause the local-depth of the bucket has remained 1. And, any bucket having a local depth lessthan the global depth is pointed-to by more than one directory.*

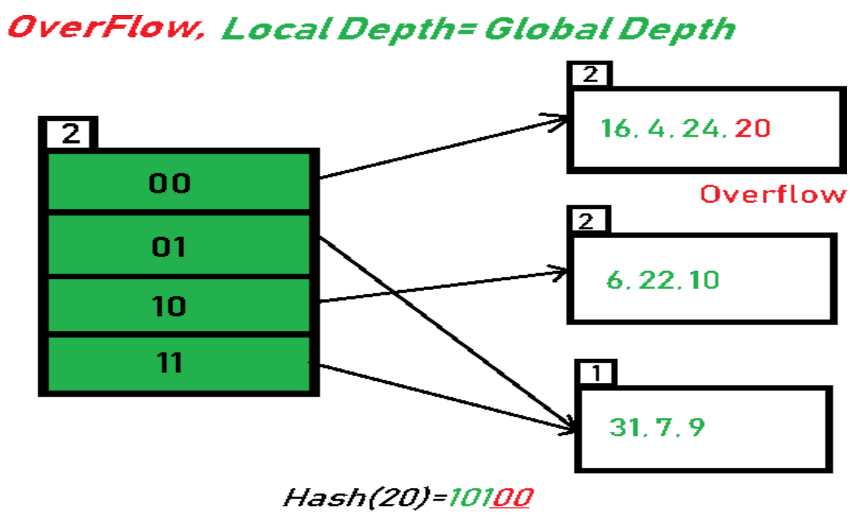
Inserting 24 and 10: 24(110**00**) and 10 (10**10**) can be hashed based on directories with id 00 and 10. Here, we encounter no overflow condition.



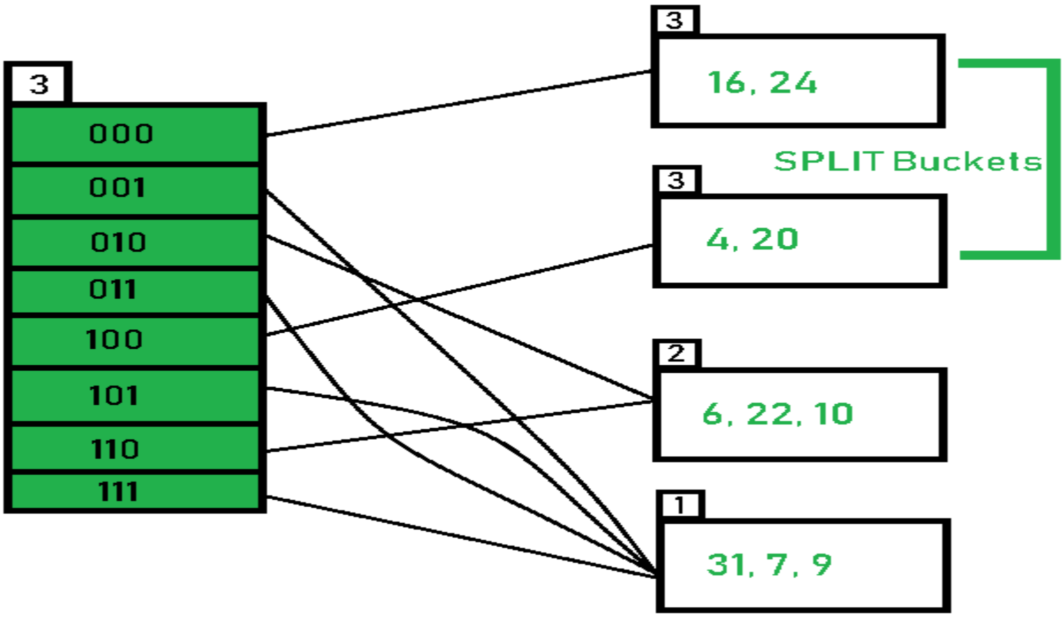
Inserting 31,7,9: All of these elements[31(111**11**), 7(1**11**), 9(10**01**)] have either 01 or 11 in their LSBs. Hence, they are mapped on the bucket pointed out by 01 and 11. We do not encounter any overflow condition here.



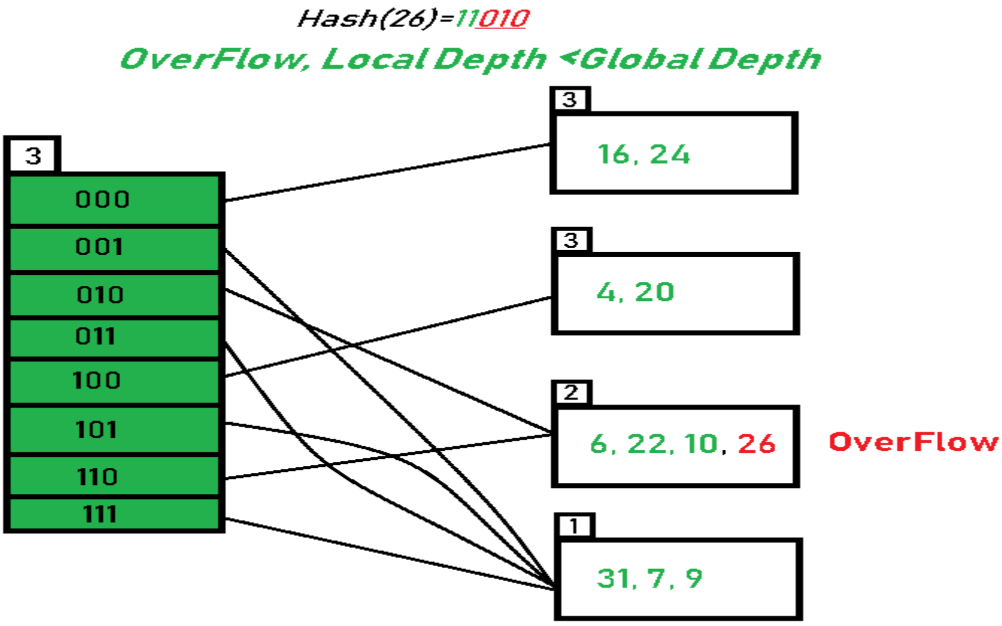
Inserting20: Insertionofdataelement20(101**00**)willagaincausetheoverflowproblem.



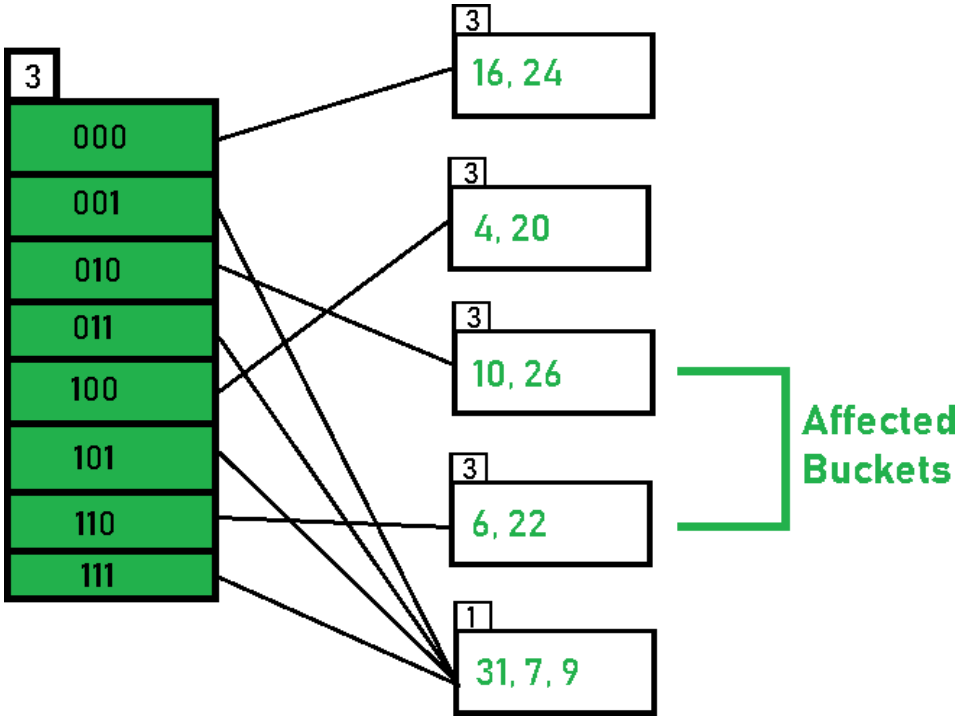
20isinsertedinbucketpointedoutby00.Asdirectedby**Step7-Case1**,sincethe **localdepth of the bucket = global-depth**, directory expansion (doubling) takes place along with bucket splitting. Elements present in overflowing bucket are rehashed with the new global depth. Now, the new Hash table looks like this:



Inserting 26: Global depth is 3. Hence, 3 LSBs of 26(11**010**) are considered. Therefore 26 best fits in the bucket pointed out by directory 010.



Thebucketoverflows, and,asdirected by **Step7-Case 2,** sincethe **localdepthofbucket< Global depth (2<3)**, directories are not doubled but, only the bucket is split and elements are rehashed. Finally,theoutputofhashingthegivenlistofnumbersisobtained.



- **Hashingof11Numbersisthuscompleted.**

Advantages

Dataretrievalisless expensive(interms ofcomputing).

No problem of Data-loss since the storage capacity increases dynamically.

With dynamic changes in hashing function, associatedold values are rehashed w.r.t the new hash function.

LimitationsofExtendibleHashing:

The directory size may increase significantly if several records arehashed on the same directory while keeping the record distribution non-uniform.

Size of every bucket isfixed and this method is complicated to code

Memory is wasted in pointers when the global depth andlocal depth differencebecomes drastic.

Example-2:consideraprominentexampleofhashingthefollowingelements:

26,14,16,12,14,10,21,17,19,20.

BucketSize:2(Assume)

HashFunction: SupposetheglobaldepthisX.ThentheHashFunctionreturnsXLSBs.

Solution:

First,calculatethebinaryformsofeachofthegivennumbers.

26-11010

14-01110

16-10000

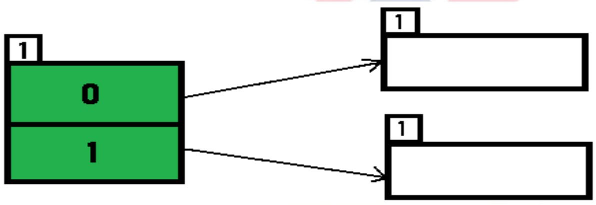
12-01100

10-01010

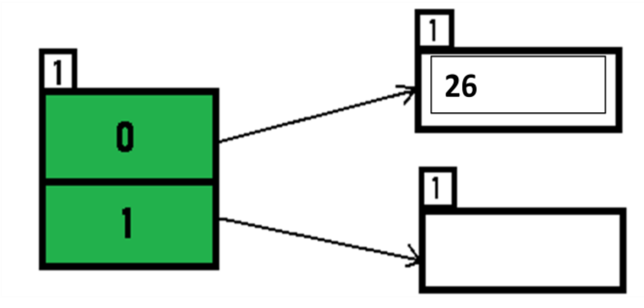
21-10101

17-10001

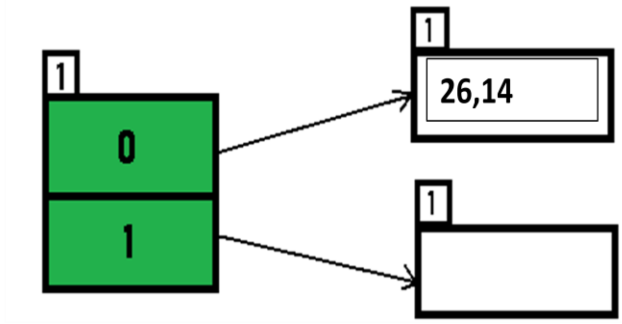
- Initially, the global-depth and local-depth is always 1. Thus, the hashing frame looks like this:



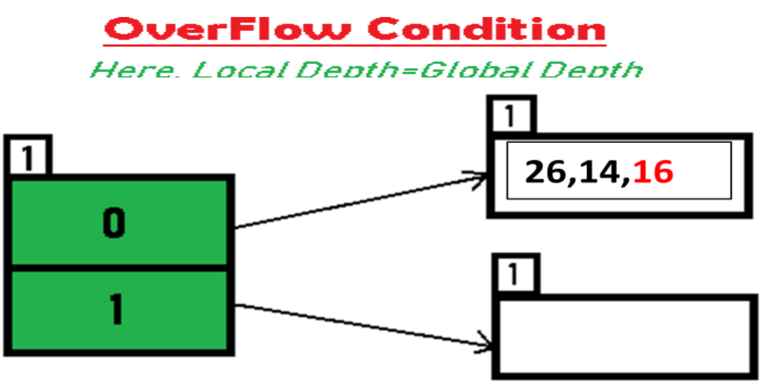
Inserting26:



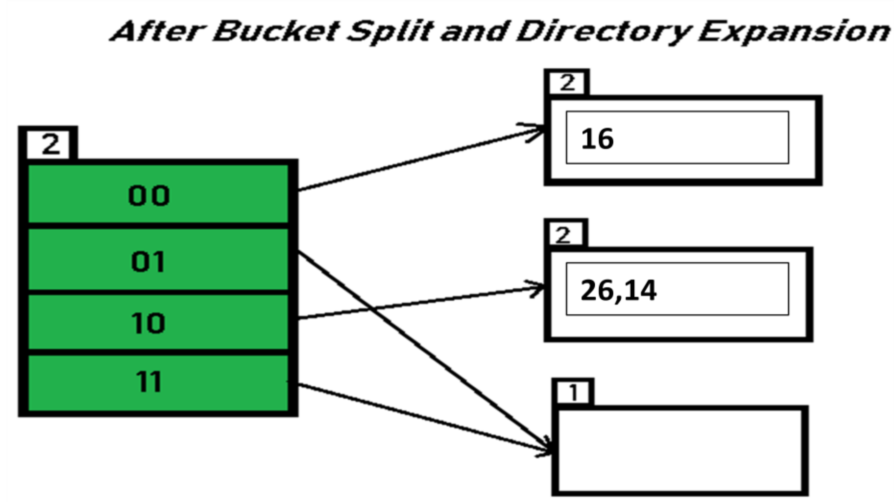
Inserting14:



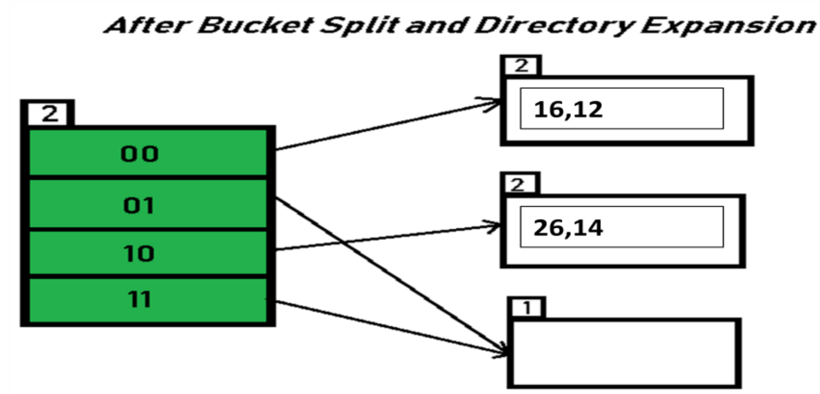
Inserting16:



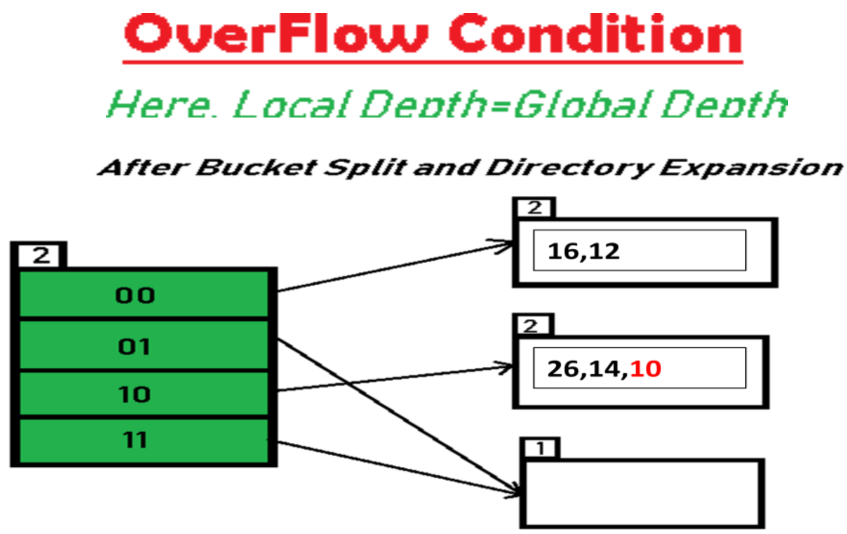
As directed by **Step 7-Case 1**, Since Local Depth = Global Depth, the bucket splits and directory expansion takes place. Also, rehashing of numbers present in the overflowing bucket takes place after the split. And, since the global depth is incremented by 1, now, the global depth is 2. Hence, 26,14,16 are now rehashed w.r.t 2 LSBs.[26(110**10**),14(0111**0**),16(**10000**)] .



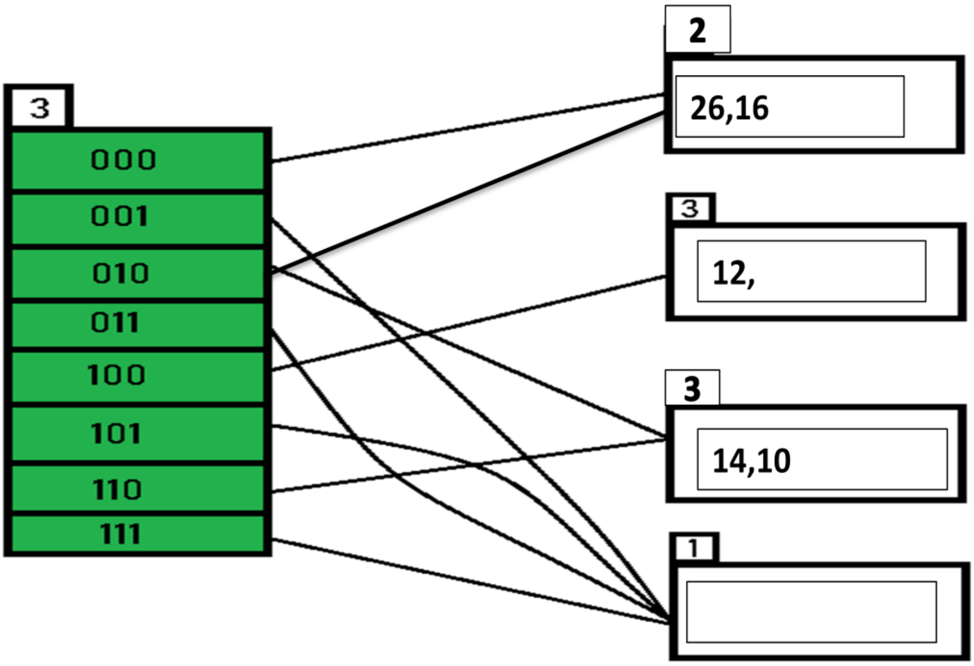
Inserting12:



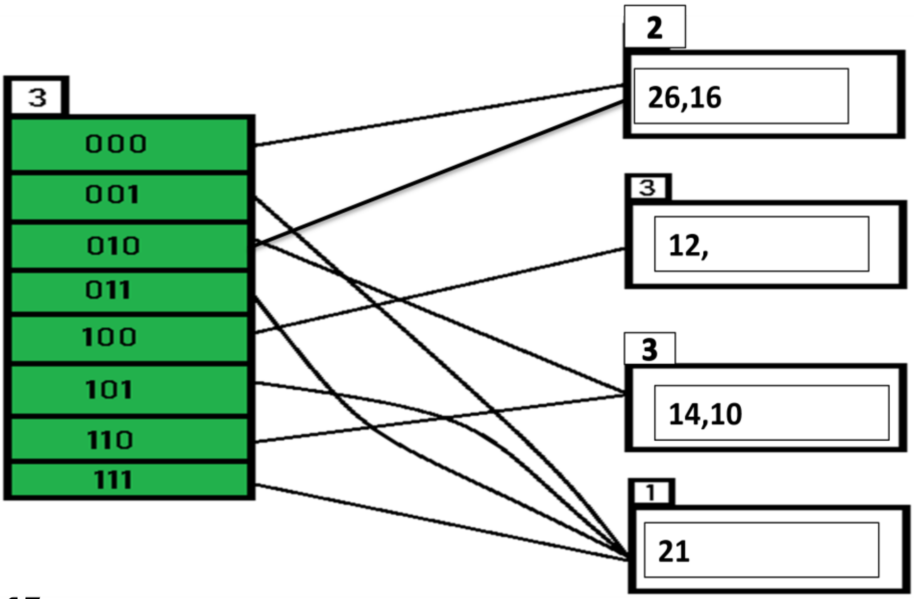
Inserting10:



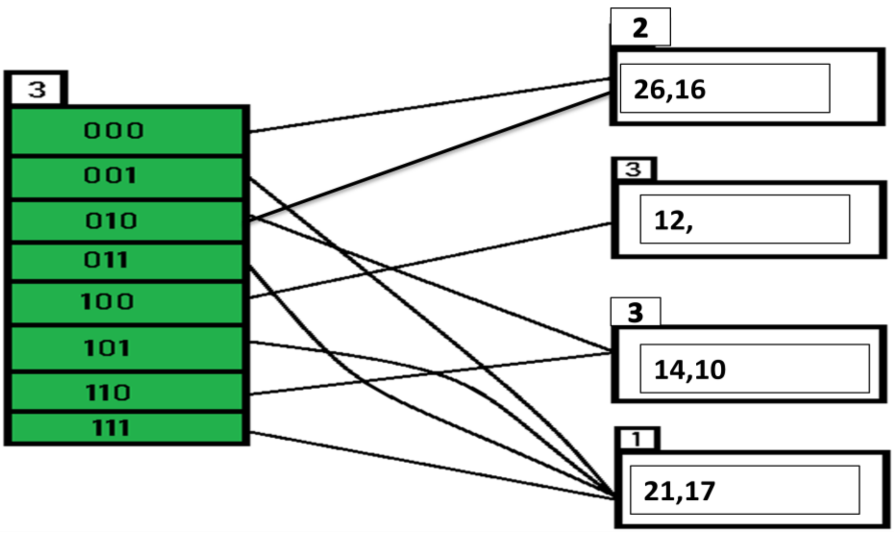
10isinsertedinbucketpointedoutby**10**.Asdirectedby **Step7-Case1**,sincethe **localdepth of the bucket = global-depth**, directory expansion (doubling) takes place along with bucket splitting. Elements present in overflowing bucket are rehashed with the new global depth. Now, the new Hash table looks like this:



Inserting



Inserting17:



Insertioniscompleted

ImplementationofDictionaries:

In computerscience,adictionaryisanabstractdatatypethatrepresentsanorderedor unordered list of key-value pair elements where keys are used to search/locate the elements inthe list. In a dictionary ADT, the data to be stored is divided into two parts:

- **Key**
- **Value.**

Each item stored in a dictionary is represented by a key-value pair. Key is used to access the elements in the dictionary. With the key we can access value which has more information about the element.

CharacteristicsofDictionary

- **Key-Value Pairs:**Dictionaries store data as key-value pairs where each key is unique and maps to exactly one value.
- **DirectAccess:**Theprimaryfeatureofdictionariesistoprovidefastaccesstoelementsnot bytheirposition,asinlists orarrays,butbytheirkeys.
- **DynamicSize:**Likemanyabstractdatatypes,dictionariestypicallyallowfordynamic resizing. New key-value pairs can be added, and existing ones can be removed.
- **Ordering:**Somedictionariesmaintaintheorderofelements,suchasorderedmapsor sorted dictionaries. Others, like hash tables, do not maintain any particular order.
- **Key Uniqueness:**Each key in a dictionary must be unique, though different keys can map to the same value.

Typesof Dictionary

Therearetwomajorvariationsof dictionaries:

Ordereddictionary.

- Inanordereddictionary,therelativeorderisdeterminedbycomparisononkeys.
- Theordershouldbecompletelydependentonthekey.

Unordereddictionary.

- Inanunordereddictionary,noorderrelationis assumedonkeys.
- Onlyequalityoperationcanbeperformedonthekeys.

Example1:

Key	Value
FirstName	Mahesh
LastName	Babu
Address	Hyderabad
Age	45

Example2:

The results of a classroom test could be represented as a dictionary with student'snames as keys and their scores as the values:

results={"Sachin":65,"Dhoni” :70, "Kohili":55, "Irfan" :50, “Laxman":40}

BasicDictionaryOperations

The dictionary ADT provides operations for inserting the records, deleting the records and searching the records in the collection of databases. Dictionaries typically support so many operations such as:

- **Insert(x,D)**->insertionofelementx(key&value)intodictionaryD.

Insert(key, value)

Example:Insert(age, 40)

- **Delete(x,D**)->deletionofelementx(key&value)fromthedictionaryD.

delete(key)

Example:delete(age)

- **Search(x,D)**->searchingtheprescribedvalueofxinthedictionaryDwithakeyofan element x.

search(key)–value

Example:search(age)–40

- **Member(x,D)**->Itreturns“true” ifxbelongstoDelsereturns“false”.
- **size(D)**->It returnsthecountoftotalnumberofelementsindictionaryD.
- **Max(D)**->ItreturnsthemaximumelementinthedictionaryD.
- **Min(D)**->ItreturnstheminimumelementinthedictionaryD.

Example: Consideranemptyunordereddictionaryandthefollowingsetofoperations:

Operation	Dictionary	Output
insertItem(5,A)	{(5,A)}	
insertItem(7,B)	{(5,A), (7,B)}	
insertItem(2,C)	{(5,A), (7,B), (2,C)}	
insertItem(8,D)	{(5,A), (7,B), (2,C), (8,D)}	
insertItem(2,E)	{(5,A), (7,B), (2,C), (8,D), (2,E)}	
findItem(7)	{(5,A), (7,B), (2,C), (8,D), (2,E)}	B
findItem(4)	{(5,A), (7,B), (2,C), (8,D), (2,E)}	NO_SUCH_KEY
findItem(2)	{(5,A), (7,B), (2,C), (8,D), (2,E)}	C
size()	{(5,A), (7,B), (2,C), (8,D), (2,E)}	5
removeItem(5)	{(7,B), (2,C), (8,D), (2,E)}	A
removeAllItems(2)	{(7,B), (8,D)}	C, E
findItem(4)	{(7,B), (8,D)}	NO_SUCH_KEY