Collision Resolution Techniques

Problem: Employee Management System

Suppose, we have a company with 20 Employees. Each employee is assigned a 5 digit Employee ID, which is used to search the employee from the company's employee file.
You need to store the information effectively in the system so that we can easily apply the CRUD operations on the data.

Problem: Employee Management System

1	Employee_Name	EmpID	Salary	Position	DOB	Gender	MaritalStatus	DateofHire	EmploymentStatus	Department	ManagerName	RecruitmentSource	Engagement	EmpSatisfaction	Absences
2	Adinolfi, Wilson	10026	62506	Productioi	####	M	Single	7/5/2011	Active	Production	Michael Albert	LinkedIn	4.6	5	1
3	Ait Sidi, Karthikey	10084	1E+05	Sr. DBA	####	M	Married	3/30/2015	Voluntarily Termina	IT/IS	Simon Roup	Indeed	4.96	3	17
4	Akinkuolie, Sarah	10196	64955	Productioi	####	F	Married	7/5/2011	Voluntarily Termina	Production	Kissy Sullivan	LinkedIn	3.02	3	3
5	Alagbe,Trina	10088	64991	Productioi	####	F	Married	1/7/2008	Active	Production	Elijiah Gray	Indeed	4.84	5	15
6	Anderson, Carol	10069	50825	Productioi	####	F	Divorced	7/11/2011	Voluntarily Termina	Production	Webster Butler	Google Search	5	4	2
7	Anderson, Linda	10002	57568	Productioi	####	F	Single	1/9/2012	Active	Production	Amy Dunn	LinkedIn	5	5	15
8	Andreola, Colby	10194	95660	Software E	####	F	Single	#########	Active	Software Eng	Alex Sweetwater	LinkedIn	3.04	3	19
9	Athwal, Sam	10062	59365	Productioi	####	M	Widowed	9/30/2013	Active	Production	Ketsia Liebig	Employee Referral	5	4	19
10	Bachiochi, Linda	10114	47837	Productioi	####	F	Single	7/6/2009	Active	Production	Brannon Miller	Diversity Job Fair	4.46	3	4
11	Bacong, Alejandr	10250	50178	IT Support	####	M	Divorced	1/5/2015	Active	IT/IS	Peter Monroe	Indeed	5	5	16
12	Baczenski, Racha	10252	54670	Productioi	####	F	Married	1/10/2011	Voluntarily Termina	Production	David Stanley	Diversity Job Fair	4.2	4	12
13	Barbara, Thomas	10242	47211	Productioi	####	M	Married	4/2/2012	Voluntarily Termina	Production	Kissy Sullivan	Diversity Job Fair	4.2	3	15
14	Barbossa, Hector	10012	92328	Data Anal	####	M	Divorced	#########	Active	IT/IS	Simon Roup	Diversity Job Fair	4.28	4	9
15	Barone, Francesc	10265	58709	Productioi	####	M	Single	2/20/2012	Active	Production	Kelley Spirea	Google Search	4.6	4	7
16	Barton, Nader	10066	52505	Productioi	####	M	Divorced	9/24/2012	Voluntarily Termina	Production	Michael Albert	On-line Web applica	5	5	1
17	Bates, Norman	10061	57834	Productioi	####	M	Single	2/21/2011	Terminated for Cau	Production	Kelley Spirea	Google Search	5	4	20
18	Beak, Kimberly	10023	70131	Productioi	####	F	Married	7/21/2016	Active	Production	Kelley Spirea	Employee Referral	4.4	3	16
19	Beatrice, Courtne	10055	59026	Productioi	####	F	Single	4/4/2011	Active	Production	Elijiah Gray	Google Search	5	5	12
20	Becker, Renee	10245	1E+05	Database	####	F	Single	7/7/2014	Terminated for Cau	IT/IS	Simon Roup	Google Search	4.5	4	8
21	Becker, Scott	10277	53250	Productioi	####	M	Single	7/8/2013	Active	Production	Webster Butler	LinkedIn	4.2	4	13

Hashing: Employee Management System

However, in cases where the keys are large and cannot be used directly as an index, we should use Hashing.

In hashing, large keys are converted into small keys by using hash functions. The values are then stored in an Array data structure called hash table.

Hashing: Collisions

Since a hash function gets us a small number for a key which is a big integer or string, there is a possibility that two keys result in the same value. The situation where a newly inserted key maps to an already occupied slot in the hash table is called collision and must be handled using some collision handling technique.

Technique 1: Linear Probing

If there is a collision for the position of the key value then the linear probing technique search sequentially and assigns the next free space to the value.

10277	10061	10002	10062	10084	10242	10026	10265	10088	10069	10250	10066	10252	10012	10194	10114	10196	10023	10055	10245
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Technique 1: Linear Probing

Can you write the general formula for hash Function?

10277	10061	10002	10062	10084	10242	10026	10265	10088	10069	10250	10066	10252	10012	10194	10114	10196	10023	10055	10245
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Technique 1: Linear Probing

The general Formula for Linear Probing is:

$$h(k, i) = [h(k) + i] \bmod m$$

Where

m = size of the hash table,
$h(k) = (k \bmod m)$,
i = the probe number that varies from 0 to $m-1$.

Technique 1: Linear Probing

What is the disadvantage/issue with Linear Probing?

Technique 1: Linear Probing

What is the disadvantage/issue with Linear Probing?

One of the problems with linear probing is **Primary clustering**, many consecutive elements form groups and it starts taking time to find a free slot or to search for an element.

Technique 2

What is the Solution to avoid Primary Clustering?

Technique 2: Quadratic Probing

What is the Solution to avoid Primary Clustering?
There is another technique that says just update your general formula a little bit and now instead of linearly finding the next empty slot you find it by quadratically.

Technique 2: Quadratic Probing

The general Formula for Quadratic Probing is:

$$h(k, i) = [h(k) + i^2] \bmod m$$

Where

m = size of the hash table,
$h(k) = (k \bmod m)$,
i = the probe number that varies from 0 to $m-1$.

Hashing with Quadratic Probing

Input: 10062

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002		10084		10026		10088	10069					10194		10196			

Hashing with Quadratic Probing

Input: 10062

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084		10026		10088	10069					10194		10196			

Hashing with Quadratic Probing

Input: 10114

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084		10026		10088	10069					10194		10196			

Hashing with Quadratic Probing

Input: 10114

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084		10026		10088	10069					10194	10114	10196			

Hashing with Quadratic Probing

Input: 10250

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084		10026		10088	10069					10194	10114	10196			

Hashing with Quadratic Probing

Input: 10250

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084		10026		10088	10069	10250				10194	10114	10196			

Hashing with Quadratic Probing

Input: 10252

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084		10026		10088	10069	10250				10194	10114	10196			

Hashing with Quadratic Probing

Input: 10252

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084		10026		10088	10069	10250		10252		10194	10114	10196			

Hashing with Quadratic Probing

Input: 10242

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084		10026		10088	10069	10250		10252		10194	10114	10196			

Hashing with Quadratic Probing

Input: 10242

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084		10026		10088	10069	10250	10242	10252		10194	10114	10196			

Hashing with Quadratic Probing

Input: 10012

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084		10026		10088	10069	10250	10242	10252		10194	10114	10196			

Hashing with Quadratic Probing

Input: 10012

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084		10026		10088	10069	10250	10242	10252	10012	10194	10114	10196			

Hashing with Quadratic Probing

Input: 10265

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084		10026		10088	10069	10250	10242	10252	10012	10194	10114	10196			

Hashing with Quadratic Probing

Input: 10265

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084	10265	10026		10088	10069	10250	10242	10252	10012	10194	10114	10196			

Hashing with Quadratic Probing

Input: 10066

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084	10265	10026		10088	10069	10250	10242	10252	10012	10194	10114	10196			

Hashing with Quadratic Probing

Input: 10066

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084	10265	10026	10066	10088	10069	10250	10242	10252	10012	10194	10114	10196			

Hashing with Quadratic Probing

Input: 10061

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		10002	10062	10084	10265	10026	10066	10088	10069	10250	10242	10252	10012	10194	10114	10196			

Hashing with Quadratic Probing

Input: 10061

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	10061	10002	10062	10084	10265	10026	10066	10088	10069	10250	10242	10252	10012	10194	10114	10196			

Hashing with Quadratic Probing

Input: 10023

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	10061	10002	10062	10084	10265	10026	10066	10088	10069	10250	10242	10252	10012	10194	10114	10196			10023

Hashing with Quadratic Probing

Input: 10055

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	10061	10002	10062	10084	10265	10026	10066	10088	10069	10250	10242	10252	10012	10194	10114	10196			10023

Hashing with Quadratic Probing

Input: 10055

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
10055	10061	10002	10062	10084	10265	10026	10066	10088	10069	10250	10242	10252	10012	10194	10114	10196			10023

Hashing with Quadratic Probing

Input: 10245

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
10055	10061	10002	10062	10084	10265	10026	10066	10088	10069	10250	10242	10252	10012	10194	10114	10196			10023

Hashing with Quadratic Probing

Input: 10245

HashFunction = EmployeeID % 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
10055	10061	10002	10062	10084	10265	10026	10066	10088	10069	10250	10242	10252	10012	10194	10114	10196			10023

Where should it be placed?

Technique 2: Quadratic Probing

This is the major disadvantage of Quadratic Probing. That is not all of the hash table slots will be on the probe sequence.

Technique 2: Quadratic Probing

This is the major disadvantage of Quadratic Probing. That is not all of the hash table slots will be on the probe sequence.

This is what we call **Secondary Clustering.** That is two records only have the same collision chain (Probe Sequence) if their initial position is the same.

Technique 3:

Although Quadratic Hashing resolved the Primary Clustering Issue, But the issue of Secondary Clustering arised.

What is the Solution to avoid Secondary Clustering?

Technique 3: Double Hashing

Although Quadratic Hashing resolved the Primary Clustering Issue, But the issue of Secondary Clustering arised.
What is the Solution to avoid Secondary Clustering?
There is another technique that says just update your general formula a little bit more and now instead of quadratically finding the next empty slot you find it by another Hash Function.

Technique 3: Double Hashing

The general Formula for Double Hashing is:

$$h(k, i) = [h_1(k) + i * h_2(k)] \bmod m$$

Where

m = size of the hash table,
$h_1(k) = (k \bmod m)$,
$h_2(k) = (PRIME + (k \bmod PRIME))$,
i = the probe number that varies from 0 to $m-1$.

Technique 3: Double Hashing

- The value returned by h_2 must never be zero (or M) because that will immediately lead to an infinite loop as the probe sequence makes no progress.

- However, a good implementation of double hashing should also ensure that all locations can be probed. For that we have to choose the Table size equal to the prime number.

Load Factor

A critical influence on performance of these techniques is the load factor; that is, the proportion of the slots in the array that are used.

Load Factor

=

No. of Elements in the Hash Table/Size of Hash Table

Load Factor

As the load factor increases towards 100%, the number of probes that may be required to find or insert a given key rises dramatically. Once the table becomes full, probing algorithms may even fail to terminate.

Rehashing

Basically, when the load factor increases to more than its pre-defined value (default value of load factor is 0.75), the complexity increases. So to overcome this, the size of the array is increased (doubled) and all the values are hashed again and stored in the new double sized array to maintain a low load factor and low complexity.

Collision Resolution Techniques

All the previous techniques store the keys inside the Hash Table therefore, these techniques are categorized as Closed Hashing.

Collision Resolution Techniques

All the previous techniques store the keys inside the Hash Table therefore, these techniques are categorized as Closed Hashing. And the elements are stored in different addresses of the array therefore, these techniques are also known as Open Addressing Techniques.

Collision Resolution Techniques

Chaining Technique uses linked lists (memory outside the hash table) to resolve the collision therefore it is called Open Hashing.

Open VS Closed Hashing

Closed Hashing (Open Addressing)	Open Hashing
Linear Probing	Chaining
Quadratic Probing	
Double Hashing	
Hopscotch hashing	
Robin Hood hashing	
Last-come-first-served hashing	
Cuckoo hashing	

Learning Objective

Students should be able to categorize the Collision Resolution Techniques