ANKESH KUMAR
NUID - 002208893

Homework 3

Problem 1

Approach to the solution

Started with calculating the Location in m=11 hash table corresponding to each Key.

Used C++ programming and here I got output in terms of the hash table function.

For each key value, following was the location in the hash table –

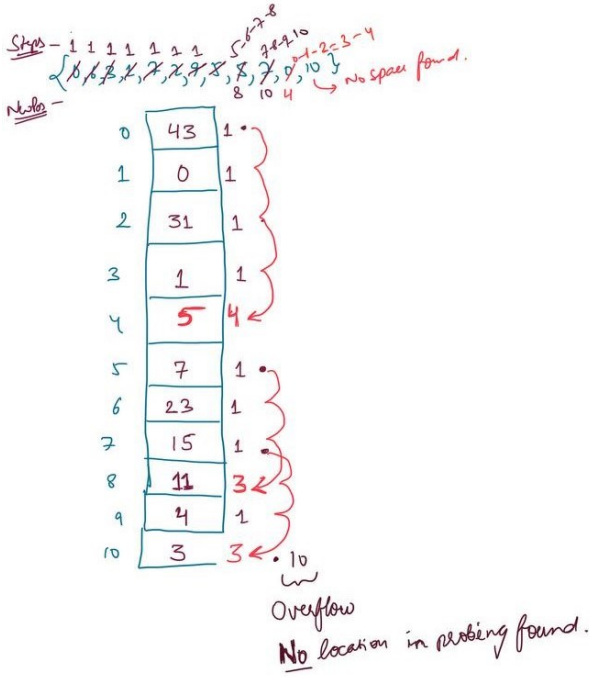
{0, 6, 3, 1, 7, 2, 9, 5, 5, 7, 0, 10}

Now, as per the placement technique,

Utilised following probing sequence –

1. For the first key placement at location, it will be placed at specified location as given above.
2. If there is any another key where the hash table index value collides, shifted the value to next index till found empty slow.
3. If all the slots were filled, then would mention it as overflow and program would give “out of memory” type error.

Rough Implementation –



ANKESH KUMAR
NUID - 002208893

Implementation

Key Value	Probe Sequence
43	0
23	6
1	3
0	1
15	7
31	2
4	9
7	5
11	5 → 6 → 7 → 8
3	7 → 8 → 9 → 10
5	0 → 1 → 2 → 3 → 4
9	10 → 0 → 1 . . . . → 9 → Overflow

	Final Hash Table Contents
0	43
1	0
2	31
3	1
4	5
5	7
6	23
7	15
8	11
9	4
10	3

ANKESH KUMAR
NUID - 002208893

Problem 2.

Approach to the problem –

Followed the instruction provided for the doing all the steps.

And corresponding hash table values were calculated.

For a. Used logic as - mm for range (0,12), dd in range (0,28) and year in range (0,4) was generated and then combined using this formula – (mm*10000+dd*100+yy) which gave 1000 random dates between Jan 1, 2000 to Dec 31, 2004.

For b and c, the code is built in program.

Hash table logic is defined in function and the main function passes the size, hash table size and the 1000 distinct array dates to the function `calculateAndDisplayCollisions` where all calculation related to the max, min, mean and variance was done based on the m value of the hash table being passed.

Here I utilised the dynamic memory allocation for array, and it was deleted at the end.

Following was the arrays generated for the count of collision happening on each index of the array –

- First two (m=97 and m=98) was distributed consistently and all the slots were occupied.
- When we had m = 100, only first 4 slot of the hash table was utilised while chaining.

```
260904 10104 61002 100904 80304 71102 200701 190603 270303 210302 130301 100502 30600 11003 100504 271100 71004 111202 170203 30202 3
1200 60803 190901 30204 10401 190903 150603 61004 130303 171002 240300 240303 210804 250701 281000 40404 90800 201104 80503 231100 60
104 250600 30804 240203 191103 220303 81000 251204 230402 31102 191200 280704 20804 10603 110302 150402 80503 190203 40401 160500 260
301 91101 160603 100502 220804 130904 121101 40904 270301 120504 211004 10304 50903 250204 50202 40804 131003 240200 90902 270104 270
401 190901 270300 220303 110201 10400 180603 151203 190901 280902 170304 150604 170102 10701 160404 70203 10703 211100 90102 20203 31
001 110502 160503 100102 191100 140202

Collisions for table size 97:
8 10 12 8 5 8 9 4 5 8 11 8 7 12 11 11 17 13 3 14 9 11 9 16 7 7 1 7 14 6 12 11 15 10 7 12 10 8 10 11 8 12 10 8 14 25 12 16 11 12 12 15
 15 12 8 10 10 6 15 15 12 10 10 10 16 19 15 8 8 10 3 7 17 10 9 15 10 6 9 10 12 10 6 10 14 9 9 13 10 10 14 5 6 6 11 11 7
Maximum collisions (subtracting 1 to account for initial occupancy): 24
Minimum collisions (subtracting 1 to account for slots without collisions): 0 (no collisions or empty slots)
Mean number of collisions per slot (including empty slots): 9.30928
Variance of collisions across all slots: 13.6982

Collisions for table size 98:
6 6 11 16 12 10 10 5 10 6 21 11 12 8 19 8 23 9 21 11 18 9 14 10 11 8 11 15 16 13 20 12 18 12 14 9 11 6 15 7 9 9 11 5 10 11 6 8 6 7 10
 12 15 11 13 6 12 3 12 5 5 7 10 9 11 3 10 6 11 7 7 9 14 7 13 8 6 9 10 8 13 3 10 5 7 5 15 7 18 5 13 6 12 7 8 18 4 9
Maximum collisions (subtracting 1 to account for initial occupancy): 22
Minimum collisions (subtracting 1 to account for slots without collisions): 2
Mean number of collisions per slot (including empty slots): 9.20408
Variance of collisions across all slots: 18.2441

Collisions for table size 100:
185 191 209 215 200 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Maximum collisions (subtracting 1 to account for initial occupancy): 214
Minimum collisions (subtracting 1 to account for slots without collisions): 184
Mean number of collisions per slot (including empty slots): 9.95
Variance of collisions across all slots: 1907.02

N:\Cprograms\VSPrograms\HW3_Sol2\Debug\HW3_Sol2.exe (process 12076) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debug
ging stops.
Press any key to close this window . . .
```

Next, about the calculations done (rounded to 2 decimal places),

	M=97	M=98	M=100
Maximum value (without collision)	24	22	214
Minimum value (without collision)	0	2	184
Mean	9.30	9.20	9.95
Variance	13.69	18.24	1907.02

ANKESH KUMAR
NUID - 002208893

Seeing on how the hast table size affects,

As the size of m is 100, when divided by the yy which only goes till 00 to 04, the mod m will always range from 0,1,2,3,5 indexes and it will chain on these locations only.

Here are two things which matter, 1. The size of m or hash table index and the 2. Type of data.

Due to both reason we can say this thing.

Variance shows that the Collision rate for the m=97 and 98 is near to uniformly spread while the collision for the m=100 is highly concentrated.

Also, there is difference in m=97 and m=98 where, 97 is prime and 98 is non-prime.Prime number has less variance and more spread and hence it is a good choice. While m=100 being the bad choice.

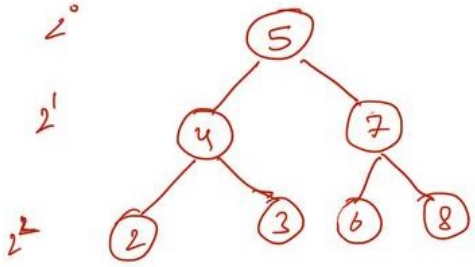
ANKESH KUMAR
NUID - 002208893

Problem 3

Binary Tree

$i = 5$

5



Parent $= \lfloor i/2 \rfloor$

Left(i) $\rightarrow 2i$

Right(i) $\rightarrow 2i+1$

Any node at index i, parent node is previous one.

$\therefore$ Parent is at index less than i

Tree is complete ie, prev nodes are filled before

Nodes at level n filled before $n+1$

Hence, parent of any node must be at index $(i/2)$.

2 node at level n

$\therefore$ 1 node at level $n-1$

$\therefore$ Parent node is calculated as $\frac{i}{2}$ where i is index.

Left & Right

ANKESH KUMAR
NUID - 002208893

∴ Parent node is calculated as $\frac{i}{2}$ where i is index.

Left & Right

From above diagram,

we can see that all the i $2i$ $2i+1$

Left values will be stored in multiple of 2^n, $2+2^n$, $2.2+2^n$ sequence while 'right' value will be stored in 2^n+1, 2^n+3, 2^n+5 etc.

∴ Left – 2^n, 2^n+2, $2^n+2.2$ - - -

Right – 2^n+1, 2^n+3, 2^n+5 . - - -

Hence the above mentioned will be

Example Root at position '1'

The root's first child goes to pos $2\times1 = 2$

Root's second child goes to position $2\times1+1 = 3$

For parent → $\lfloor\frac{3}{2}\rfloor = 1$ ← Position of parent

2×1 ← Position of Left child

$2\times1+1$ ← Position of Right child.

≈ $\lfloor i/2 \rfloor$ parent

$2\times i$ Left

$2i+1$ Right