CP2410 Practical 08

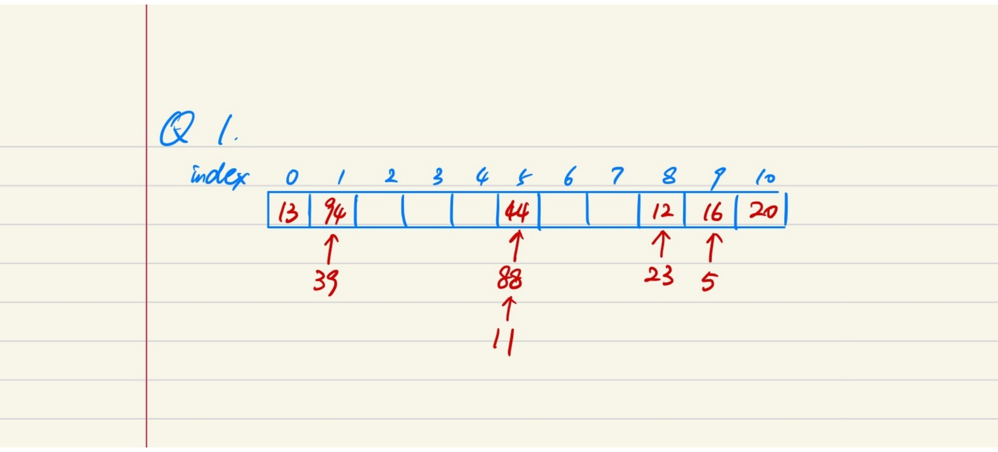
Sihan Chen, jcu ID: 14187662

Question 1

Hash function: $h(i) = (3i + 5) \mod 11$

Key	Hash Value
12	8
44	5
13	0
88	5
23	8
94	1
11	5
39	1
20	10
16	9
5	9

and handling collisions with chaining will give us a hash table below:



Question 2

If the collisions are handled by linear probing, we will have a hash table look like this:

Q2.

index	0	1	2	3	4	5	6	7	8	9	10
	13	94	39	16	5	44	88	11	12	23	20

Question 3

If the collisions are handled by quadratic probing

$$A[h(i) + j^2] \text{ (mod N)}, for\ j = 0, 1, \cdots, N - 1$$

we can build a simple python function helps us to check quadratic probing new values

```
>>> def qh(n):
...     for i in range(11):
...             print(i, ((3*n)+5+i^2)%11)
...
```

and insert elements up to the point of failure, we will have a hash table look like this (next page):

Q3.

index	0	1	2	3	4	5	6	7	8	9	10
	13	94	39	11		44	88	16	12	23	20

88 collides at 5,

j	h(i)
1	6

23 collides at 8,

j	h(i)
1	9

11 collides at 5,

j	h(i)
1	6
2	9
3	3

39 collides at 1,

j	h(i)
1	2

16 collides at 9.

j	h(i)
1	10
2	2
3	7

5 collides at 9.

j	h(i)
1	10
2	2
3	7
4	3
5	1
6	1
7	3
8	7
9	2
10	10

Since we can not get a hash value of 4, the hash table fails to insert 5

Question 4

If the collisions are handled by double hashing

$$H(i) = (h(i) + j * h'(i)) \bmod N, \text{for } j = 0, 1, \cdots, N-1$$

we can build a simple python function helps us to check quadratic probing new values

```
>>> def sec_h(n):
...     return 7 - (n%7)
...
>>> def h(n):
...     return (3*n + 5) % 11
...
>>> def dh(n):
...     for i in range(11):
...             print(i, (h(n) + i * sec_h(n))%11)
```

the final hash table would look like this:

Q4.

index	0	1	2	3	4	5	6	7	8	9	10
	13	94	23	88	39	44	11	5	12	16	20

88 collides at 5,

j	h(i)
1	8
2	0
3	3

23 collides at 8,

j	h(i)
1	2

11 collides at 5,

j	h(i)
1	8
2	0
3	3
4	6

39 collides at 1,

j	h(i)
1	4

5 collides at 9.

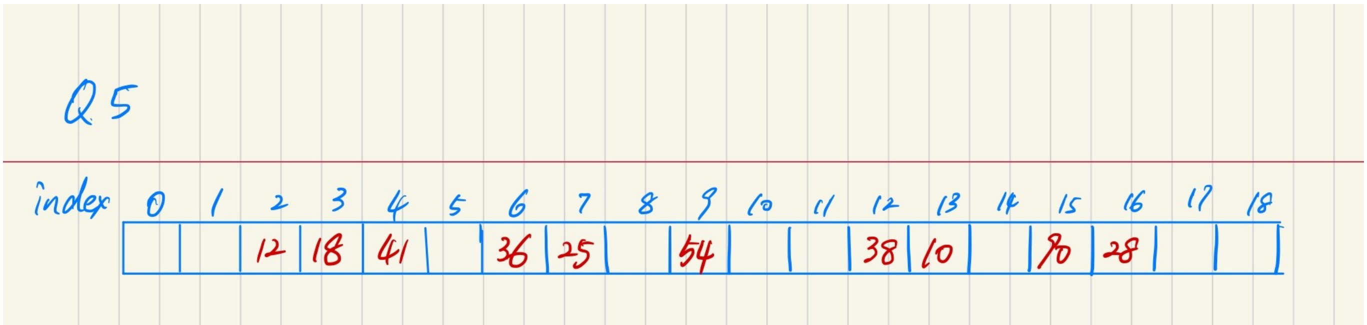
j	h(i)
1	0
2	2
3	4
4	6
5	8
6	10
7	1
8	3
9	5
10	7

Question 5

With new hash function $h(k) = 3k \mod 17$

Key	Hash Value
54	9
18	3
10	13
25	7
28	16
36	6
38	12
41	4
12	2
90	15

So, the new hash table would look like this:



Question 6

We can use the `binary search` technique to reduce search time on each row from $O(n)$ to $O(\lg(n))$. For each row of the matrix, we will split the row in middle, and keep doing the splitting while there are 0s in the left or 1s in the right (by checking last element in left and first element in right). If the left's last element is 0, we call function recursively on left half, otherwise on the right half. And the base case is length of input list is 1.

```
def count_ones(A:list[list[int]]) -> int:
    """ Return number of 1s in the matrix(n x n). """
    result = 0
    for row in A:
        result += count_ones_in_row(row)
    return result

def count_ones_in_row(row:list[int]) -> int:
    """ Return number of one in a row. """
    result = 0
    n = len(row)

    if n == 1:
        return 1 if row[0] == 1 else 0

    left = row[:n//2]
    right = row[n//2:]
    if left[-1] == 0 or right[0] == 1:
        if left[-1] == 0:
            return count_ones_in_row(left)
        else: # 1 in right
            return len(left) + count_ones_in_row(right)
    else:
        return len(left)
```

This way, we can achieve time complexity of $O(n * \lg(n))$ .And here we can see the result appears to be correct on test input:

prac08.md M q6.py U

Week08 > q6.py > ...

```python
def count_ones(A:list[list[int]]) -> int:
    """ Return number of 1s in the matrix(n x n). """
    result = 0
    for row in A:
        result += count_ones_in_row(row)

    return result

def count_ones_in_row(row:list[int]) -> int:
    """ Return number of one in a row. """
    n = len(row)

    if n == 1:
        return 1 if row[0] == 1 else 0

    left = row[:n//2]
    right = row[n//2:]
    if left[-1] == 0 or right[0] == 1:
        if left[-1] == 0:
            return count_ones_in_row(left)
        else: # 1 in right
            return len(left) + count_ones_in_row(right)
    else:
        return len(left)

A = [[1, 1, 1, 1, 1], [1, 1, 1, 0, 0], [1, 1, 0, 0, 0], [0, 0, 0, 0, 0], [1, 0, 0, 0, 0]]
print(count_ones(A))
```

PROBLEMS OUTPUT TERMINAL GITLENS SQL CONSOLE DEBUG CONSOLE

```
pwsh ~\Dev\cp2410\Week08  main +2 ~2
> python .\q6.py
11
pwsh ~\Dev\cp2410\Week08  main +2 ~2
>
```