Problems

March 2, 2017

1 Problem 1:

1.1 Implement separate chaining, a collision resolution technique. Also, discuss time complexity of each function.

```
In [ ]: class HashTable(object):
            def __init__(self, size):
                self.keys = [None] * size
                self.values = [None] * size
                self.size = size
            # Class function to be used by each object in the same manner
            def hash_key():

            def hash_function(self, value):
                key = self.hash_key(value)
                return key % self.size
            def insert(self, value):
                key = self.hash_function(value)
```

2 Problem 2: Pascal's triangle is important for calculating the binomial cofficients. Pascal's triangle generated coefficients are also useful in Probability systems. How would you generate pascal's triangle given the number of rows that are required?

2.1 Print Pascal's triangle of n rows:

```
In [64]: def pascals_triangle(n, rows=[list()]):
             if n == 1:
                 rows = [[1]]
                 return [1]
             elif n == 2:
                 rows = [[1], [1, 1]]
                 return [1, 1]
             else:
                 rows = [[1], [1, 1]]
```

```
            while n > 2:
                row_n_minus_1 = rows[-1]
                # the new row will have size bigger than previous row by 2
                row_n = [0] * (len(row_n_minus_1) + 1)
                for i in xrange(len(row_n)):
                    if i == 0:
                        row_n[i] = row_n_minus_1[i]
                    elif i == len(row_n) - 1:
                        row_n[i] = row_n_minus_1[i-1]
                    else:
                        row_n[i] = row_n_minus_1[i-1] + row_n_minus_1[i]
                rows.extend(list(list([row_n])))
                n = n - 1
            return rows
```

```
In [66]: pascals_triangle(16)
```

```
Out[66]: [[1],
          [1, 1],
          [1, 2, 1],
          [1, 3, 3, 1],
          [1, 4, 6, 4, 1],
          [1, 5, 10, 10, 5, 1],
          [1, 6, 15, 20, 15, 6, 1],
          [1, 7, 21, 35, 35, 21, 7, 1],
          [1, 8, 28, 56, 70, 56, 28, 8, 1],
          [1, 9, 36, 84, 126, 126, 84, 36, 9, 1],
          [1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1],
          [1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1],
          [1, 12, 66, 220, 495, 792, 924, 792, 495, 220, 66, 12, 1],
          [1, 13, 78, 286, 715, 1287, 1716, 1716, 1287, 715, 286, 78, 13, 1],
          [1, 14, 91, 364, 1001, 2002, 3003, 3432, 3003, 2002, 1001, 364, 91, 14, 1],
          [1,
           15,
           105,
           455,
           1365,
           3003,
           5005,
           6435,
           6435,
           5005,
           3003,
           1365,
           455,
           105,
           15,
           1]]
```

In [ ]: